Thèse de Doctorat

Francisco Gamboa Quintanilla

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes

Sous le label de l'Université Nantes Angers Le Mans

Discipline :
Spécialité : Automatique et Informatique Appliquée
Laboratoire : Institut de Recherche en Communications et Cybernétique de Nantes IRCCyN

En vue d'une soutenance le 25 Novembre 2015

École doctorale : Sciences et technologies de l'information et mathématiques
Thèse N°

Couplage des Architectures Holonique et Orientée-Services pour la Conception de Systèmes de Production Agiles.

JURY

Rapporteurs :	**Trentesaux, Damien,** Professeur des Universités à l'ENSIAME, Université de Valenciennes et du Hainaut-Cambrésis, LAMIH
	Borangiu, Theodor, Professeur à l'University Politehnica of Bucharest, Roumanie, CIMR
Examinateurs:	**L'Anton, Anne**, Maître de conférences à l'IUT de Nantes, Université de Nantes, IRCCyN
	Chacòn, Edgar, Professeur à l'Université des Andes, Mérida, Venezuela, LaSDAI
Directeur de thèse :	**Castagna, Pierre**, Professeur à l'IUT de Nantes, Université de Nantes, IRCCyN
Co-encadrant :	**Cardin, Olivier**, Maître de conférences à l'IUT de Nantes, Université de Nantes, IRCCyN

Sommaire

Résumé

Pour atteindre des objectifs en termes de réactivité, flexibilité, réduction des coûts et augmentation de la productivité, les entreprises sont à la recherche de solutions remplaçant les systèmes de contrôle manufacturiers conventionnels afin de gagner en robustesse, adaptabilité, configuration rapide et maximisation de l'utilisation des ressources. Les Architectures Holoniques et Orientées-Services ont été proposées en tant que solutions de conception pour de tels types de systèmes, le premier dans le domaine manufacturier fournissant de la flexibilité au niveau contrôle, le dernier en informatique au niveau processus. La combinaison de ces deux architectures a été reconnue comme une solution encore plus attractive pour la conception des systèmes de production de prochaine génération. .

Cependant, l'intégration du concept de services nécessite de nouveaux modèles pour s'adapter au contexte manufacturier et lui octroyer de la flexibilité, particulièrement pour la planification et l'ordonnancement au niveau processus.

Ce travail propose une architecture pour Système de Production Holonique Orienté-Services (Service Oriented Holonic Manufacturing System (*SoHMS*), piloté autour des produits, en tant que méthodologie pour modéliser les systèmes de production flexibles en utilisant un ensemble de modèles et méthodologies pour sa spécification. Au niveau des modèles, notre contribution se porte sur un modèle de service adapté à la production, un modèle flexible de processus basé sur les Réseaux de Petri, un protocole d'orchestration et des stratégies pour l'ordonnancement et la planification intégrés de ces processus (Integrated Process Planning and Scheduling) afin d'explorer efficacement les solutions offertes tout en évitant une explosion combinatoire, et des modèles et protocoles holoniques permettant de former une architecture *SoHMS*. Au niveau méthodologique, nous proposons un framework permettant de concevoir des services de production et des spécifications de processus, la création d'une ontologie de services d'application et une méthodologie pour modéliser les gammes sous forme de Réseaux de Petri.

Le *SoHMS* résultant est un système ayant la possibilité de créer des plans de production flexibles adaptables en fonction des aléas survenant sur le système, ayant la capacité d'intégrer rapidement et facilement des nouvelles ressources et compétences et sachant proposer des solutions faisables de production avant l'arrivée aux points de décision correspondants.

Introduction

Les systèmes de production doivent aujourd'hui être plus performants, que ce soit en termes de rendement ou en termes de qualité de service. Ceci pour répondre à des demandes du marché de plus en plus variables et personnalisées. Ainsi, on parle aujourd'hui de « *Mass Customization* », concept alliant l'idée de production de masse, pour satisfaire un grand nombre de clients, et de personnalisation du produit pour chaque client en particulier. Cela entraine un enjeu majeur pour les entreprises qui, dans un contexte économique difficile, doivent être flexibles pour s'adapter à cette personnalisation, et agiles pour rester compétitives. Elles doivent être capable de se reconfigurer rapidement et à moindre coût, pour s'adapter à des mutations de plus en plus rapides, dues à des évolutions des marchés, des procédés de production et des besoins des clients. Ces chalenges ont conduit la communauté scientifique à explorer de nouvelles approches du pilotage des systèmes de production.

Ainsi, les initiatives de recherches françaises (MRT, SPI-SHS, PROSPER, …) et internationales IMS (Intelligent Manufacturing Systems), HMS (Holonic Manufacturing Systems) ont permis de proposer des nouveaux concepts, modèles conceptuels, et architectures s'appuyant sur la décentralisation et la distribution des fonctions dites « intelligentes » de conduite des moyens et ressources, par des modèles hétérarchiques, distribués et appuyés par des systèmes et solutions multi-agents. Ces nouvelles architectures permettent des comportements beaucoup plus flexibles, permettant de réagir à des disfonctionnements ou à des évolutions des systèmes. Elles nécessitent par contre des modèles et des structures de données permettant de décrire la flexibilité du système pour connaitre, pour chaque décision devant être prise, le domaine des solutions pertinentes.

Parallèlement, dans le domaine de l'informatique, notamment dans les applications Web, on a vu depuis les années 2000 se développer un type d'architecture bénéficiant d'un grand succès pour la conception de systèmes agiles, autonomes et reconfigurables. Ce paradigme est celui des Architectures SoA : « Service-Oriented Architecture ». L'architecture orientée services (traduction généralement admise du terme SoA) est un modèle d'interaction applicative qui met en œuvre des services (composants logiciels) :

- avec une forte cohérence interne (par l'utilisation d'un format d'échange pivot, le plus souvent XML) ;
- des couplages externes « lâches » (par l'utilisation d'une couche d'interface interopérable, le plus souvent à travers des services web).

Rapidement, même si le paradigme des SoA a été conçu pour une implémentation dans des applications informatiques, la communauté scientifique a cherché à appliquer les principes et pratiques de conception d'une architecture distribuée au domaine de la production. La mise en œuvre des principes au niveau atelier (au niveau du Manufacturing Execution System – MES – plus précisément) peut potentiellement apporter les mêmes avantages d'agilité, d'autonomie et de reconfigurabilité, au système de production que les SoA dans le domaine de l'informatique.

A travers cette thèse, nous proposons de lier ces deux paradigmes prometteurs, celui de systèmes de production holonique (HMS) et celui d'architecture orientée services (SoA). Pour cela, nous proposons la

définition d'une architecture pour un Système de Production Holonique Orienté-Services (Service Oriented Holonic Manufacturing System (*SoHMS*). Nous avons donc cherché à définir successivement un modèle de service adapté à la production, puis un modèle flexible de processus basé sur les Réseaux de Petri, et enfin un protocole d'orchestration et des stratégies pour l'ordonnancement et la planification de ces processus (Integrated Process Planning and Scheduling).

Ce document est organisé en 6 chapitres. Le premier constitue, de manière générale, une introduction aux systèmes de production holonique et au SoA. Ensuite, dans le deuxième chapitre, nous nous proposons d'adapter les SoA au domaine de la production. Pour cela, nous partons de la notion de famille de produits et nous montrons comment cette notion conduit à celle de famille de processus. Nous définissons alors le concept de services de production (MServices). L'étude de la modélisation des *MServices* et de leur «*matching*» nous amène alors au paradigme de processus orientés services. Nous pouvons alors aborder le cœur de notre proposition dans le troisième chapitre : Le Système Holonique de Production Orientée Services (*SoHMS*). Nous proposons, à travers ce chapitre une méthodologie de modélisation permettant la définition des *SoHMS*. Le produit et ses processus de fabrication, dans toute leur complexité, sont au cœur de notre approche. Aussi, nous consacrons le quatrième à la définition d'un modèle de *Gammes Flexibles Orientées-Services*. Ce modèle, basé sur les réseaux de Petri permet non seulement la description des processus conduisant à obtenir le produit, mais son aspect dynamique nous permettra ensuite de représenter l'état d'avancement des processus. Ce modèle dynamique sera donc utilisé dans la description de *l'Orchestration des Plans de Production Orientés-Services* qui fait l'objet du cinquième chapitre. Le sixième et dernier chapitre est consacré à la mise en œuvre de nos propositions à travers un cas concret.

Chapitre 1

Les Architectures Distribuées : Flexibilité au Niveau de la Commande

Ce premier chapitre a pour but l'analyse des caractéristiques des architectures de contrôle de la littérature permettant d'augmenter intrinsèquement la flexibilité disponible dans le système de production au niveau de la commande. L'étude de chaque architecture a pour objectif la compréhension de la manière avec laquelle les éléments du système sont identifiés et modélisés, ainsi que les différentes interactions mises en place entre les éléments.

Le chapitre commence avec une description de l'évolution des architectures de commande des systèmes distribues au cours des dernières décennies, qui ont vu l'apparition d'architectures alternatives classées dans la catégorie des architectures hétérarchiques. Le développement récent de la thématique des Systèmes Contrôlés par le Produit a accompagné les évolutions matérielles permettant d'ouvrir le réseau d'information à un plus grand nombre d'éléments mobiles à moindre coût. En parallèle, le paradigme des systèmes de productions holoniques au sein de la communauté producticienne et les architectures orientées-services dans le domaine de la conception d'architecture en informatique se sont développées en parallèle et montrent un potentiel intéressant dans chacun de leur domaine de prédilection. Les dernières sections de ce chapitre présentent un court état de l'art de chacun de ces concepts.

1.1 LES ARCHITECTURES DE PILOTAGE CLASSIQUES

1.1.1 Les Architectures Hiérarchiques

L'approche hiérarchique est basée sur une perception naturelle des systèmes complexes où la partition des ordres de haut niveau en ordres plus spécifiques et de portée plus limitée ne facilite pas seulement leur applicabilité mais aussi leur compréhension. Dans ces systèmes comme dans les organisations sociales humaines, des ordres se transmettent du haut vers le bas et les informations sur la progression de ces ordres ou de l'état du système du bas vers le haut pour générer de nouvelles prises de décision et de nouveaux ordres. Les niveaux les plus hauts se basent sur l'hypothèse d'un comportement prévisible de tous les composants (dans la hiérarchie et dans l'environnement) pour la prise de décision. À cause de ceci et au fait que ces architectures nécessitent une structure fixe lors de leur exécution, ils possèdent certains désavantages :

- Système difficile à modifier, au cas où l'on souhaite étendre le système ou le mettre à niveau avec une nouvelle technologie. Il faut l'arrêter, et mettre à jour toutes les structures d'information de niveaux hauts et leur influence sur le comportement des niveaux plus bas ;

- La modification des structures d'information pendant l'exécution sont quasiment impossibles.

Dû à l'hypothèse sur la nécessité d'un comportement prévisible des composants et de l'environnement, le programme de production devient rapidement invalide en présence d'aléas sur l'un de ces deux domaines. Dans certains cas, les plans de production peuvent même être invalides avant d'être calculés.

Dans cette approche très centralisée, une hiérarchie forte existe entre tous les composants du système qui s'appuie sur un flot descendant de commandes et un flot remontant d'informations (Figure 1). Ainsi, un niveau supérieur définit les contraintes et objectifs à atteindre par le niveau suivant. De ce fait, un problème de coordination entre les différents niveaux apparaît, puisqu'il est très fréquent que les problèmes de chaque niveau soient interdépendants. La prise de décision peut porter sur un horizon temporel représentant la durée de validité/applicabilité des décisions.

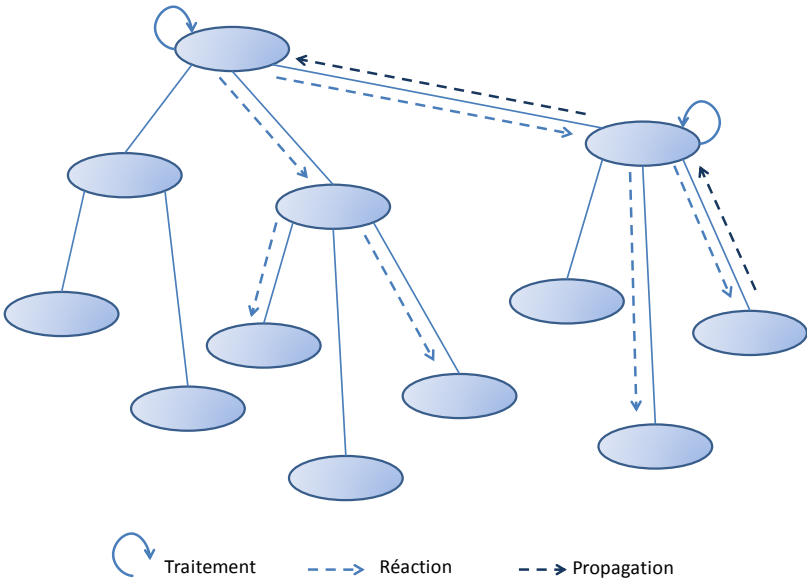


Figure 1 Topologie d'architecture hiérarchique (Blanc, 2006)

1.1.2 Les Architectures Hétérarchiques

L'approche hétérarchique a été conçue pour résoudre ce problème d'incapacité à gérer les aléas induits par le système centralisé. Cette architecture est composée d'entités intelligentes, qui peuvent alternativement représenter une ressource dans le système ou bien une tâche du processus. Dû à l'apport d'intelligence locale à ces agents, ils ont la capacité d'adaptation à des situations qui n'ont pas été envisagées auparavant, grâce à des règles de comportement qui leur ont été programmées.

Dans cette approche très décentralisée, la notion de maitre-esclave trouvée dans le système hiérarchique est de fait plus souple, la relation n'existant pas entre tous les éléments et pouvant évoluer dans le temps (Figure 2). Ainsi, les flux d'informations et de décisions sont multiples et peuvent également évoluer selon l'évolution du système.

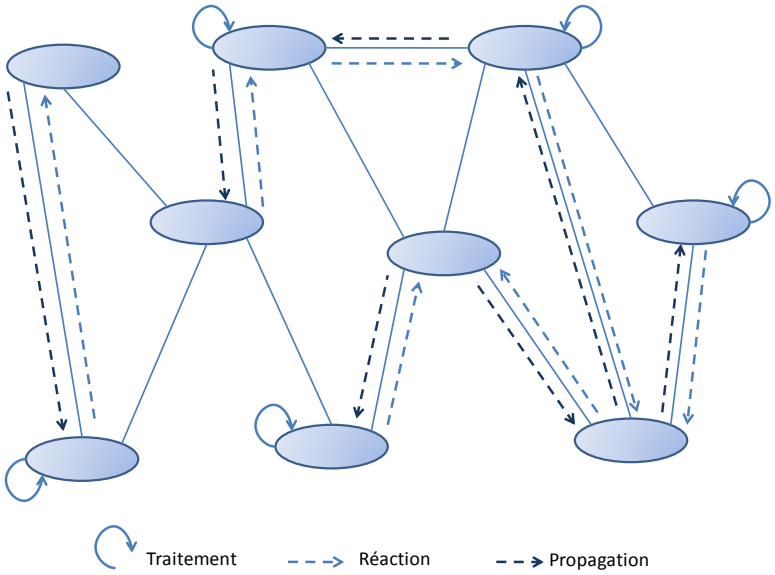


Figure 2 Structure hétérarchique de contrôle (Blanc, 2006)

Les ordres et informations sont échangés via un protocole de négociation entre les entités. Le protocole de négociation le plus connu est le protocole Contract-Net (Borangiu et al., 2014; Smith, 1980). En schématisant, dans le contexte d'une négociation, les participants peuvent adopter deux rôles: gestionnaires de tâches ou clients. Le contrat se fait alors entre le gestionnaire et le client généralement sur un système d'enchères. En vue de cette négociation, basée sur des informations mises à jour en temps-réel et sans aucune connaissance à priori du comportement des autres agents, simplement des offres qu'ils fournissent, ces systèmes deviennent tolérants aux aléas, tout en étant toujours capables de trouver une solution faisable, mais généralement pas optimale.

Les avantages principaux de ce type d'architecture proviennent de la décentralisation de l'information et surtout de la prise de décision. Ainsi, il est possible de retarder au maximum cette prise de décision afin de bénéficier au mieux de la mise à jour en temps réel des données de l'atelier. Ces systèmes possèdent toutefois aussi des désavantages importants vis-à-vis de l'accessibilité de l'information :

- L'indépendance des agents limite leur accessibilité à l'information globale sur le système. Ils ne considèrent que l'information concernant les agents avec lesquels ils interagissent directement, mais pas de ceux qui l'affectent de façon indirecte ;
- La performance globale est très sensible au réglage des règles de comportement ;
- Il n'y a pas de garantie de performance minimale si le système sort du champ de travail pour lequel les règles ont été ajustées ;
- La performance globale peut être garantie seulement autour d'une valeur moyenne, quand l'agent se situe dans son champ de travail ;
- Le cycle de vie d'un ordre dépend fortement de la nature et de l'état des autres ordres ;
- La gestion des aléas dans le système est effective seulement s'il existe des alternatives.

1.2 LES SYSTEMES CONTROLES PAR LE PRODUIT : UNE PERSPECTIVE DE COMMANDE

L'évolution du marché vers des produits de plus en plus personnalisables et une évolution dans les pratiques de la production comme l'émergence des systèmes hétérarchiques demandent et poussent les concepteurs de ces dernières à adopter de nouveaux paradigmes, ou plutôt de nouvelles manières de regarder l'organisation de la production. Ce nouveau paradigme (nouveau quant à sa présence industrielle) est le paradigme des *Systèmes Contrôlés par le Produit (SCP).* Ce développement est une évolution logique à un contexte où le produit devient de plus en plus individualisé et où la production n'est plus fortement liée à la notion de lot. Dans cette section, on présentera un court état de l'art sur ce paradigme.

1.2.1 Les technologies d'auto-identification

Le pilier fondamental du paradigme des SCP est l'association des produits aux données relatives à son statut. La théorie menant à concevoir de tels systèmes était et reste très attractive, mais elle trouve des complications pour son implémentation, dû à un manque de technologie pouvant associer les informations au produit de manière virtuelle. Cependant, le paradigme a retrouvé son attractivité et sa viabilité grâce à l'émergence des technologies dites « *infotroniques* » (Pannequin, 2007). Grâce à ces technologies, l'intégration des objets informationnels dans les activités de prise de décision dans un système de production est devenue possible. Parmi ces technologies, on peut mentionner les étiquettes RFID (pour Radio Frequency Identification), la communication Bluetooth ou la communication Zigbee.

La technologie RFID est celle qui a trouvé un développement le plus fort dans le domaine de la production. Elle consiste en la création d'étiquettes électroniques, sur lesquelles on peut effectuer de la lecture ou de l'écriture de données au travers d'émissions par radio fréquence. Cette action peut se faire de manière locale, lorsque l'étiquette est présentée à un point de contrôle. D'autre part, les technologies Bluetooth et Zigbee sont deux technologies de communication sans fil développées à fin des années 90 par les sociétés Ericsson et Motorola respectivement. La technologie Bluetooth est conçue pour donner de l'interopérabilité à des produits mobiles permettant des connexions de type ad-hoc de quelques mètres à presque 100m selon la classe. Zigbee pour sa part ajoute à la liste des objectifs la basse consommation et se présente comme un standard ouvert. De plus, les connexions Zigbee sont de type Broadcast et non ad-hoc. (Baker, 2005) présente un comparatif de ces deux technologies dans le cade de leur utilisation dans des applications industrielles.

Ces technologies permettent l'identification des produits et leur association aux informations de la production. Cela apporte aux systèmes de production des capacités augmentées de traçabilité de produits, surtout importantes dans la production personnalisée et l'implémentation des stratégies de production en flux tiré.

1.2.2 Perspectives de contrôle

La réactivité de ces systèmes se base sur la prise des décisions en temps réel pendant la production, en se basant sur des règles établies à priori. L'idée derrière est de déplacer le moment de la prise de décision au plus proche du moment de leur application. Cette approche suggère de donner un rôle plus important au produit, passant d'un simple amas de matière circulant au sein du système à un acteur dans la prise de décision, capable d'interagir avec d'autres composants du système. Selon (Pannequin, 2007), dans un tel contexte, le produit devient un *Objet Nomade de Production,* défini comme « un objet capable de fournir des données aux clients, coopérer avec les autres entités participant à son évolution, adapter ses règles de

décision en fonction de l'expérience acquise durant les phases précédentes de son cycle de production et coordonner des processus de décisions ».

Avec l'association de la matière et des informations, les systèmes de production évoluent d'un flux de matière et d'informations découplés vers un flux de produits, associant informations et matière à la fois. De telles nouvelles perspectives conduisent à repenser la manière d'organiser et coordonner la prise de décision du système de contrôle vers une organisation où le produit devient le coordinateur de sa production. Ceci implique le passage d'une organisation hiérarchique et intégrée du pilotage vers une organisation permettant de l'interopérabilité et de l'intelligence, postulant le produit comme un vecteur de coordination et le pilote des ressources (Morel et al., 2007; Pannequin, 2007). Un produit délégant de telles responsabilités suggère le besoin de *Produits Intelligents.*

1.2.3 Qu'est-ce qu'un Produit Intelligent ?

Selon la définition de (Kärkkäinen, 2003) la notion de produit intelligent est associée à la gestion du cycle de vie de chaque produit, de manière individuelle, en intégrant le flux physique et informationnel afin d'offrir des services en utilisant un réseau. On trouve une autre définition dans (Ramirez, 2006) décrivant un produit intelligent comme un objet porteur de données, pointeur vers un système d'information, fournisseur et demandeur de services et comme un objet sensitif et enrichi avec des capteurs et des actionneurs.

Une définition du point de vue fonctionnel, probablement la plus référencée dans le cadre de la production et de la « *supply chain* » est celle de (Wong et al., 2002). Un produit intelligent y est défini comme un objet avec une partie physique et une partie informationnelle dotée de capacités de stockage d'information, de communication, d'action et de décision. Une classification en deux niveaux d'intelligence selon les qualités d'interaction du produit est proposée. Un produit de *niveau 1* est doté de capacités à s'identifier, stocker des informations et communiquer avec l'environnement passivement. Ce niveau est considéré comme *orienté-information,* où l'activité des produits se résume à donner et garder les informations pour que le système puisse décider de leur traitement. Un produit *niveau 2* possède toutes les propriétés d'un niveau 1 plus la capacité à dialoguer avec son environnement et la capacité de participer à la prise de décisions du système. Ils sont considérés comment *orientés-décision.*

L'application d'une intelligence de *niveau 1* autant qu'une de *niveau 2* implique plusieurs avantages pour le pilotage du système. Avec l'augmentation du produit à travers une intelligence de *niveau 1*, on peut profiter de la capacité de traçabilité pour tous les produits lors de sa production, en temps réel, jusqu'au niveau composant. On gagne également la capacité de détection des goulots (au sens de la gestion de production) en temps réel pour réagir en concordance, et plus important (surtout pour les produits complexes qui requièrent des processus de qualité supérieure) la capacité de superviser la qualité des produits en continu permettant un bouclage des flux de matériaux ou produits pour leur retraitement. Le *niveau 2* ajoute à la liste la capacité d'auto-organisation de tous les éléments actifs du système. Grâce à cette intelligence, les produits sont capables de conserver l'état de leur production et la continuer plus tard ou sur un autre support de production, tout en identifiant les services disponibles et négociant avec les fournisseurs. Depuis la perspective du contrôle, l'application d'un *niveau 2* réalise une distribution de la commande, ce qui la rend plus facile à développer.

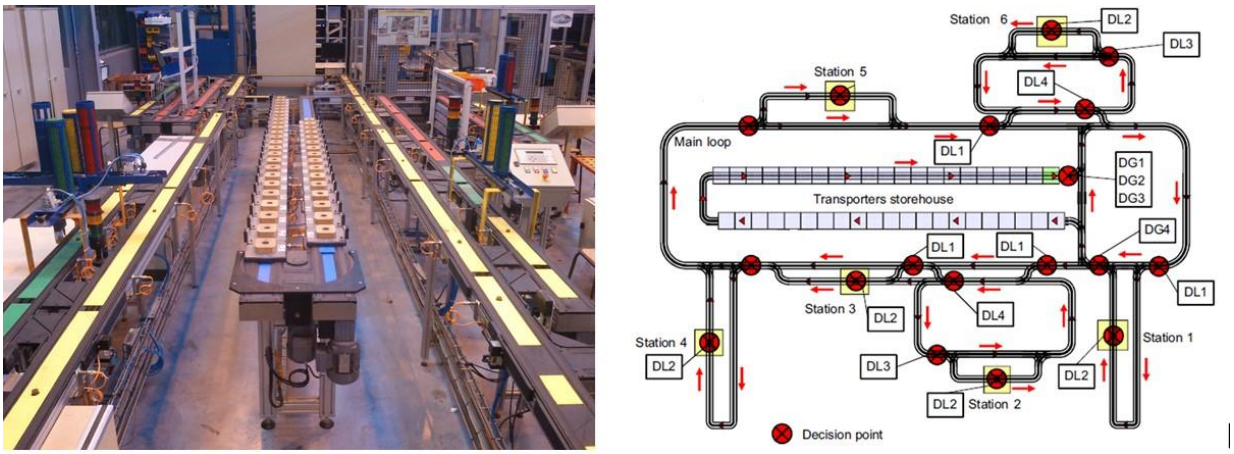


Figure 3 Système contrôlé par le produit *niveau 1* (Cardin and Castagna, 2009)

Dans (Cardin and Castagna, 2009), Figure 3 on trouve une application d'un système contrôlé par le produit avec des produits intelligents *niveau 1*. Il s'agit d'une ligne d'assemblage industrielle doté de tapis roulants et postes de service. Le pilotage se fait par cinq centres de décisions dont un contient la description du système en Grafcet. Le produit intelligent est formé par l'association d'un produit avec sa base transporteuse. Cette base est équipée avec une étiquette électronique RFID, laquelle sert d´identification à vérifier à tout point de décision dans le système. Une poursuite de ces travaux a été proposée dans (Gamboa Quintanilla et al., 2013a), qui proposent une évolution du même système vers une intelligence de *niveau 2*, en équipant la base, appelée palette, d'une architecture de communication lui permettant de communiquer en continu avec l'ensemble des acteurs du système, d'une puissance de traitement des informations basée sur un processeur Arduino et d'actionneurs pour se diriger activement au sein de la chaine de production (Figure 4).

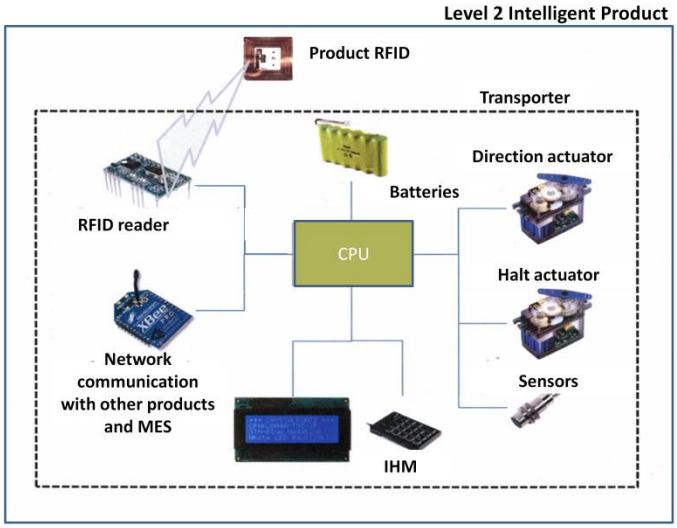


Figure 4 Produit Intelligent de *Niveau 2* (Gamboa Quintanilla et al., 2013a)

1.3 LES ARCHITECTURES HOLONIQUES

1.3.1 Motivation originelle

Pour résoudre les problèmes de pilotage des systèmes de production classiques précités, la communauté académique a notamment orienté ses recherches vers d'autres paradigmes afin de satisfaire leurs besoins de réactions aux changements dans l'environnement de la production et aux différents aléas qui peuvent survenir lors de la production. Parmi ces paradigmes, on peut mentionner la production Bionique (Ueda, 1992), l'usine fractale (Warnecke, 1993) et la production holonique. Cette dernière a rencontré le plus de succès dans le monde académique aussi bien qu'industriel (Blanc et al., 2008), car elle intègre les meilleurs attributs des contrôles hiérarchique et hétérarchique, permettant aux utilisateurs de bénéficier des avantages de ces deux approches simultanément (Trentesaux, 2009).

En effet, le manque de flexibilité des architectures avec des topologies de contrôle de type *pyramidales* et le manque de garanties et d'optimisation des architectures avec une topologie distribuée de type *plate* ont provoqué des évolutions de ces deux extrêmes. Des applications avec des topologies pyramidales ont de plus en plus ajouté d'interactions parmi les contrôleurs de même niveau, tandis que les topologies dites plates intègrent de plus en plus d'associations et d'agrégations ayant comme résultat l'émergence de topologies mixtes, comme celle présentée par (Trentesaux, 2009) illustrée Figure 5 .

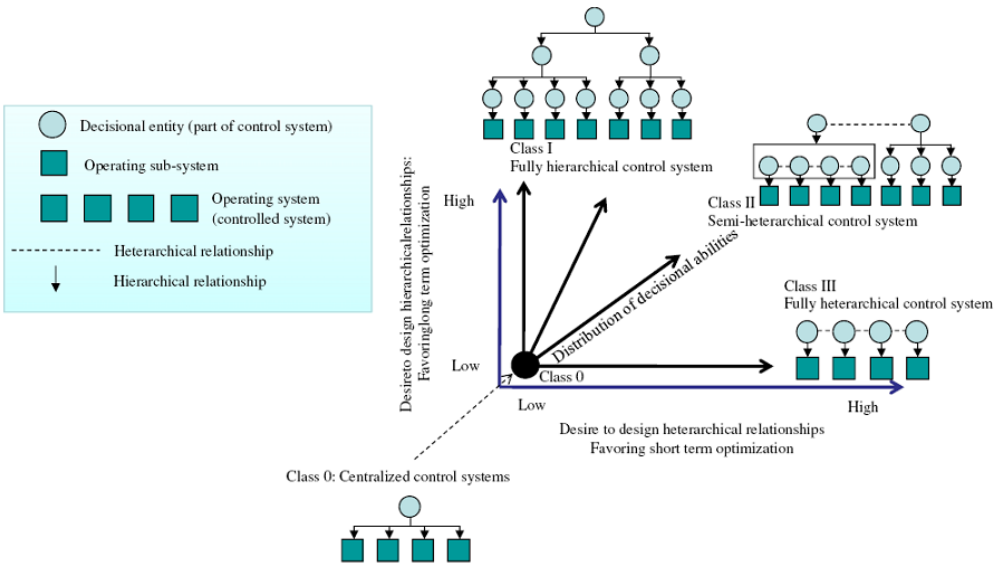


Figure 5 Migration vers une topologie de contrôle mixte (Trentesaux, 2009)

En effet, les caractéristiques complémentaires des architectures hiérarchiques et hétérarchiques, l'optimisation et la productivité d'un côté et la flexibilité et réactivité de l'autre, ont mené vers ce que l'on appelle une *holonification* (Pannequin, 2007) des architectures de contrôle, issue de l'ajout des éléments centralisés aux structures plates et à la relaxation des contraintes des relations maitre-esclave dans les structures pyramidales.

1.3.2 Le paradigme holonique

L'origine du concept holonique est issue du travail du philosophe hongrois Arthur Koestler, lors de sa tentative de caractérisation du comportement des organisations sociales des organismes vivants, tels que ceux des colonies de termites ou des colonies de fourmis (Koestler, 1968). C'est dans cette étude qu'il introduit le terme *Holon,* afin de décrire l'unité fondatrice de telles organisations. Le terme holon vient du mot grecque *holos* signifiant « entier » accolé au suffixe *on* qui suggère une partie ou une particule (comme

pour « neutron » et « proton »). La théorie de (Koestler, 1968) suggère que, dans telles organisations, il existait des formes stables intermédiaires, étant ceux-ci des systèmes simples, qui permettent leur propre évolution vers des systèmes plus complexes. Les holons dans ces organisations peuvent se comporter de deux manières différentes selon le point de vue avec lequel on les regarde : comme des « tout » depuis la perspective de ses subordonnés ou comme une « partie » du point de vue inverse. A ce propos, (Koestler, 1968) avait proposé le concept de *Holarchies*, type de structure hiérarchique non bornée vers le haut ni vers le bas, capable d'association et de dissociation.

La relation d'agrégation parmi les holons peut être décrite comme dans la Figure 6 . Dans ce diagramme, on peut observer le caractère fractal et potentiellement récursif des holons. Le *holon de référence* peut être considéré comme le holon visible depuis une perspective avec lequel on peut interagir sans avoir besoin de connaitre sa constitution.

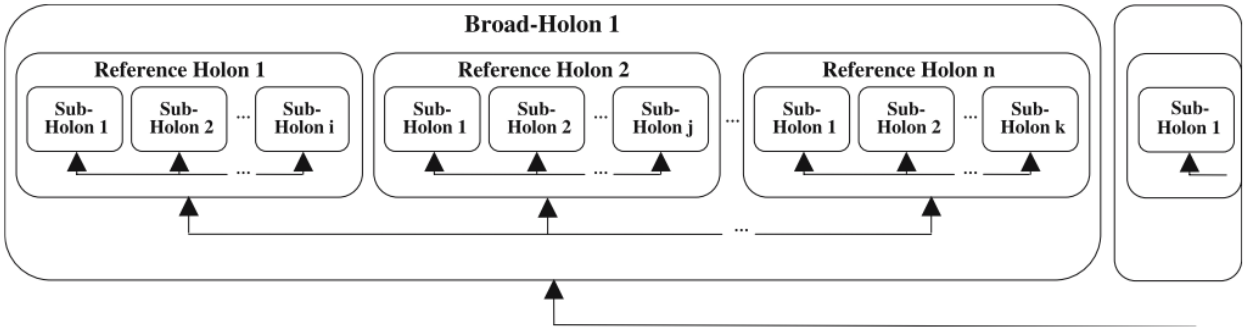


Figure 6 Structure d'agrégation holonique (Babiceanu and Chen, 2006)

Ce terme a été utilisé pour la première fois dans le contexte de la production par des chercheurs japonais dans les années 80 avec la proposition d'un contrôleur dit holonique. Ce n'est qu'à partir des années 90 que le concept holonique réussit à être considéré comme une solution pour la conception des systèmes de production capable de faire face aux nouvelles problématiques. A ce propos, un consortium de différents pays a été établi dans la cadre d'un programme IMS (Intelligent Manufacturing Systems). Ce programme, HMS, avait pour but de développer des outils et implémenter des Systèmes Holoniques de Production sur des problèmes industriels réels en cherchant à atteindre une certaine stabilité face aux perturbations, de l'adaptabilité face aux changements dans l'environnement et une utilisation efficace des ressources (Babiceanu and Chen, 2006).

Avec l'intention de guider les efforts de recherche du concept holonique appliqué à la production, le consortium HMS a établi une série des définitions importantes (Christensen, 1994; Valckenaers et al., 1994) :

> **Holon :** Entité autonome et coopérative fonctionnant comme un élément constitutif d'un système de production pour réaliser des fonctions de transport, transformation, stockage d'informations et/ou d'objets physiques. Il est constitué d'une partie informatique et/ou d'une partie physique. Un holon peut faire partie d'un autre holon et peut être formé par d'autres holons plus granulaires.
>
> **Autonomie *:*** Capacité d'une entité à planifier et exécuter ses propres plans d'action.
>
> **Coopération *:*** processus pour lequel les entités d'un système (les holons) développent et exécutent des plans d'action de manière conjointe.

Holarchie : Ensemble de holons qui coopèrent entre eux pour atteindre un objectif commun. C'est dans une holarchie que l'on définit l'ensemble des règles qui régissent le comportement des holons et qui en même temps limitent leur autonomie.

L'autonomie et la coopération sont les deux caractéristiques les plus importantes des holons (Babiceanu and Chen, 2006). C'est grâce à leur autonomie que les holons sont capables de décider de leur propre parcours en vue d'accomplir leurs objectifs, sans besoin de consulter des entités « supervisantes ». Grâce à la coopération, les holons réussissent à établir des plans communs et les exécuter ensemble.

Une des contributions les plus importantes de ce concept dans la modélisation des systèmes de production est la distinction entre contrôleurs de produits et contrôleurs de ressources (Valckenaers et al., 1994) :

Contrôleurs de Produit : en charge de la succession des opérations et paramètres du processus de production.

Contrôleurs de Ressource : en charge de l'exécution des opérations de production et/ou de transport.

Même si les HMS ne sont pas nécessairement orientés vers le produit, cette distinction donne une place potentielle pour l'application des SCP. En effet, du fait de la reconnaissance des produits comme entité active dans le système de production, cela permet de faire des HMS des architectures potentielles pour l'implémentation des SCP.

En résumé, ce qui fait le succès de cette approche est le fait qu'elle combine les meilleurs attributs des structures organisationnelles hiérarchiques et hétérarchiques (McFarlane and Bussmann, 2003). De plus, la structure holonique préservant la stabilité d'une architecture hiérarchique et la flexibilité des hétérarchies offre un contexte favorable pour le développement d'un système de contrôle distribué (Valckenaers et al., 2003) tout en gardant les avantages de la modularité et de la décentralisation.

Relations dans les systèmes holoniques

Les organisations holoniques, pour parvenir à l'établissement de plans et à coordonner leurs actions en vue de ces plans, nécessitent des méthodologies et des protocoles de communication et d'échange d'informations qui est vital pour l'accomplissement des objectifs à la fois individuels et surtout globaux. Parmi les potentielles relations holoniques qui apparaissent dans la littérature, on peut citer*: la coordination, la coopération, la collaboration, la négociation* et la *communication* (Babiceanu and Chen, 2006).

- *Coordination* : c'est l'établissement des règles de comportement ayant pour but le respect de certaines contraintes dans le comportement des holons pour ainsi éviter l'obstruction et les duplications d'efforts.
- *Coopération* : La coopération implique des interactions, volontaires ou pas, avec un (des) autre(s) holon(s) pour l'accomplissement d'objectifs communs ou pour des bénéfices communs. Parmi les outils pour l'accomplissement de la coopération, on peut mentionner dans le domaine de l'intelligence artificielle : la coalition et le regroupement de holons.
- *Collaboration* : la collaboration peut être vue comme un genre de coopération mais où l'acte de coopération est délivré. Les holons échangent des informations en vue de l'accomplissement d'un objectif commun.
- *Négociation* : Souvent utilisé dans le domaine des HMS pour faire référence à l'allocation des ressources. La négociation représente les stratégies que les holons utilisent en cas de conflits entre leurs objectifs et les objectifs d'autres entités. Une négociation peut être vue comme un

ensemble d'actions coordonnées pour mener tout ou partie du système d'un état conflictuel vers un objectif commun.

- *Communication* : la communication est le pilier supportant les interactions holoniques, car aucun accord ne peut se décider sans échange d'informations. Des méthodes de communication se trouvent sous forme de messages directs, avec des connexions pair-à-pair ou bien au travers d'un « répertoire » de données. De plus, de tels échanges de messages nécessitent la définition d'un protocole correctement formalisé : le protocole Contract-Net, développé par (Smith, 1980), est le plus connu dans le domaine et fournit des procédures cohérentes que ce soit pour l'annonce des besoins ou l'établissement des contrats satisfaisant ces besoins.

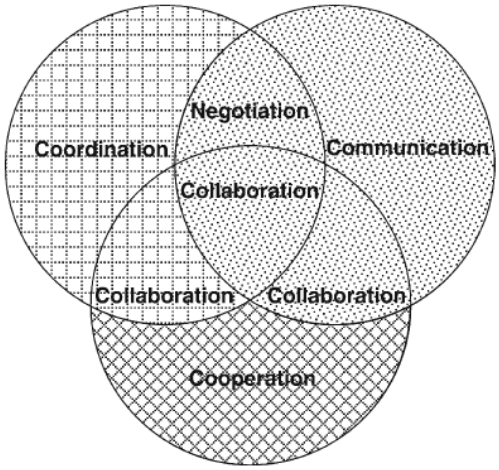


Figure 7 Diagramme de Venn : Relations Holoniques (Babiceanu and Chen, 2006)

1.3.3 Architectures de référence dans la production

Comme présenté dans la section précédente, le concept holonique donne un contexte et une terminologie pour la conception des systèmes de production holoniques HMS. Pourtant, ces concepts donnent un spectre large de possibilités en termes de modélisation des systèmes de production, d'où le besoin d'architectures de référence pour diriger ces efforts. Une architecture de référence sert à établir des principes cohérents qui permettent de structurer la conception de tels systèmes. Elle définit des concepts comme la terminologie unifiée, la structure de systèmes et les rôles et responsabilités de tous les composants qui le construisent (J. Wyns et al., 1996). Elle doit aussi faciliter la reconfiguration, l'extension et la modification du système et permettre plus de flexibilité et un espace de décision plus large pour l'interaction avec des niveaux de contrôle plus hauts (Van Brussel et al., 1998). Pour aller plus loin, une architecture de référence complète peut également donner des solutions aux interrogations suivantes (Marik and McFarlane, 2005) :

- Comment atteindre une optimisation globale du système dans système distribué ?
- Comment la structure de production évolue-t-elle pour s'adapter aux changements dans l'environnement ?
- Comment spécifier de manière formelle le comportement dynamique des systèmes holoniques ?
- Comment intégrer des stratégies d'apprentissage et d'auto-organisation ?
- Comment intégrer des ressources d'automatisation ?
- Comment développer des applications basées sur le concept holonique ?

Dans la suite de cette section, on présentera trois des architectures existantes et les plus référencées dans la communauté des HMS. On présentera une description brève de leur architecture, leur distribution des responsabilités et des informations ainsi que la description de certaines des interactions holoniques les plus intéressantes pour chaque architecture.

L'architecture PROSA

L'architecture PROSA, proposée par (Van Brussel et al., 1998), est la plus étudiée et la plus référencée dans la communauté des systèmes holoniques pour la production. Une telle architecture a été conçue comme un point de départ pour la conception et l'implémentation des systèmes holoniques du futur (Van Brussel et al., 1998). Son nom, PROSA, est conçu à partir de l'acronyme « *Product-Resource-Order-Staff Architecture* » en énumérant les différents types de holons. La conception des trois holons basiques (Produit, Ressource et Ordre) est conçue de telle manière que l'ensemble des responsabilités couvrent tous les aspects principaux et plus importantes du contrôle de la production, la planification des produits, leurs processus et la gestion des ressources d'atelier. Dans le cas des holons *Staff*, son propos est d'apporter une vision globale du système avec l'intégration d'algorithmes de planification centralisés avec l'objectif d'assister plutôt que de régir le comportement du reste des holons.

Cette classification de holons a été motivée par l'identification de trois préoccupations classiques des systèmes de production que sont :

- *Les aspects liés aux ressources* en référence aux paramètres d'utilisation des ressources. Par exemple, les vitesses optimales d'usinage ou de transport en fonction d'un certain critère.
- *Les aspects liés aux produits et les processus qu'ils subissent*. Par exemple, la gamme de production d'un produit, i.e. la séquence d'opérations à réaliser pour engendrer un produit d'une certaine qualité.
- *Les aspects de logistique,* comme par exemple la spécification des ordres de production et les dates d'échéance et priorité.

L'architecture générale formée par ces holons est modélisée dans la Figure 8 en utilisant le standard orienté-objets UML (*Unified Modeling Language*).

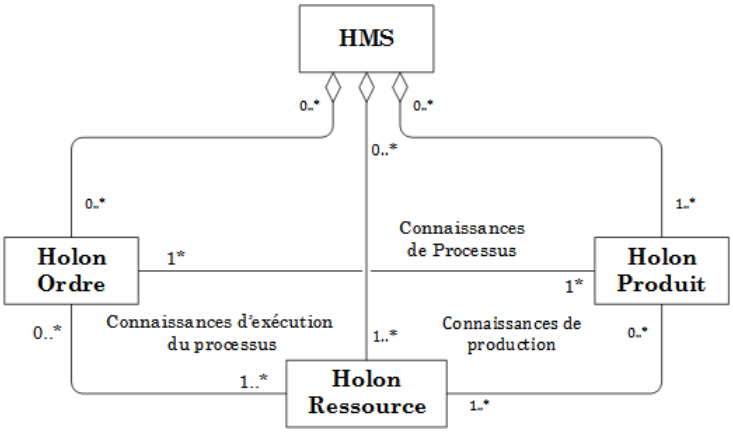


Figure 8 Architecture basique PROSA

Les Holons

Voici une description des responsabilités et fonctionnalités de chacun des acteurs d'une architecture holonique.

Le Holon Produit (HP) : Le holon produit peut-être visualisé comme un serveur d'informations sur un certain produit. Il contient toutes les informations sur le produit et les procédures qu'il doit subir pendant son cycle de vie pour atteindre un certain niveau de qualité. Exemples de ces informations sont : le bilan des matériaux, les gammes, les demandes du client, le design, les programmes de production, les procédures pour assurer la qualité, etc. Il garde le modèle du produit et non un modèle de l'état du produit.

Le Holon Ressource (HR) : Ce holon est formé d'une partie informationnelle et très souvent par une partie physique. Il peut être vu comme une abstraction de certains moyens de production voire une usine, une machine, un tapis roulant, une palette, des matériaux etc. C'est ce type de holons qui donne la capacité de production aux autres holons. Il possède les méthodes nécessaires pour l'affectation des ressources et toutes les informations et procédures pour les organiser et contrôler.

Le Holon Ordre (HO) ***:*** Il représente une tâche ou une procédure dans la séquence du processus de production. Sa principale préoccupation est de garantir la bonne exécution de cette même tâche à une date spécifique, voire avant une certaine échéance. Le HO fait la gestion de la logistique de production d'un produit en particulier. Il est souvent lié à une pièce de production avec un certain contrôle pour s'acheminer à travers le système de production.

Le Holon Staff (HS) : Ce type de holon peut être vu comme un conseiller avec un certain domaine d'expertise, donnant des informations aux autres types de holons pour qu'ils pussent prendre des décisions qui les concernent. Son existence est optionnelle dans un système de production holonique. Le HS est inspiré des organisations humaines dans lesquelles l'introduction du personnel, « staff » en anglais, a comme objectif principal la réduction de la charge et de la complexité du travail des principaux acteurs, les holons basiques. L'inclusion de ces holons permet le passage d'une architecture distribuée à une architecture partiellement hiérarchique, sans la soumission rigide des hiérarchies car la décision finale est toujours prise par le holon local. C'est grâce à cette augmentation de l'architecture par le holon staff que la robustesse et l'agilité sont découplées du système d'optimisation, s'il existe.

Le système d'optimisation est une activité complexe qui comporte une vision globale du système, et qui peut être affecté aux holons *Staff*. Portant des algorithmes centralisés, le *holon staff* donne des informations qui reflètent l'état complet du système ou bien fournit des plans d'action aux holons basiques. La Figure 9 montre un exemple d'intégration d'un *HS* ayant le rôle d'ordonnanceur. Les holons basiques prennent ces informations et essayent de les suivre au mieux dans le cas où toutes les informations sont encore valides. Par contre, si les informations données par le HS ne sont plus valides, les holons ignorent son conseil et appliquent les stratégies locales préprogrammées. Au final, le système de contrôle repasse d'une fonctionnalité hiérarchique à une hétérarchique. C'est un grand avantage lors de l'occurrence de perturbations et de changements dans le système.

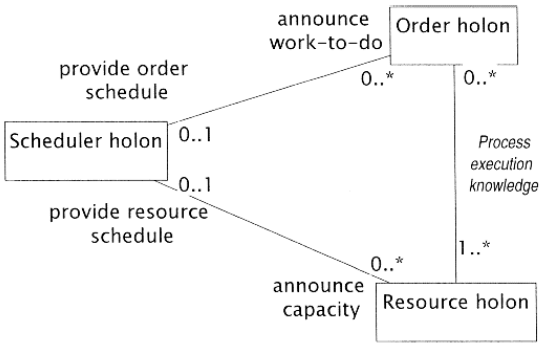


Figure 9 Intégration de l'holon *Staff* (Van Brussel et al., 1998)

Relations inter-holons

Etant donné que les responsabilités et les informations relatives à la production sont distribuées parmi les holons, le partage de ces informations implique un degré de coopération important pour la création des plans d'action et leur exécution. La Figure 10 illustre ce partage d'informations :

- *Connaissances sur le Processus* : Toute information relative à la réalisation d'un certain processus de production sur une ressource. Il s'agit des informations sur la capacité des ressources, les paramètres du processus et sa qualité.
- *Connaissances sur la Production* : il s'agit des informations relatives à la manière de produire un certain produit en utilisant certaines ressources. Par exemple : les possibles séquences d'opérations à exécuter et les méthodes pour accéder à l'information sur les programmes de production.
- *Connaissances sur l'Exécution d'une Procédure* : il s'agit d'informations concernant la progression de l'exécution de tâches sur les ressources, des informations sur comment solliciter l'initiation des taches en des ressources, comment faires des réservations, comment superviser la progression de l'exécution, comment interrompre une tâche et les conséquences de l'interruption /suspension/continuation d'une tâche.

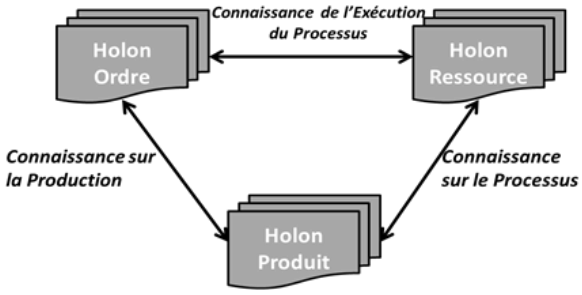


Figure 10 Partage d'informations parmi les holons basiques (Van Brussel et al., 1998)

Deux propriétés des holons de l'architecture PROSA sont les relations d'agrégation et de spécialisation. Ces propriétés rendent les architectures très flexibles et complètes pour représenter toutes les fonctions d'un système de production sans se préoccuper de sa taille. La relation d'A*grégation* permet l'association de plusieurs holons en lien direct ou indirect de manière hiérarchique, formant ainsi une holarchie qui peut être vue comme un holon en soi avec sa propre identité. Par conséquent, les holons peuvent appartenir à plusieurs agrégations. De plus, ces agrégations peuvent être de différents niveaux hiérarchiques. Cette structuration d'un des holons en sous-systèmes hiérarchiques facilite la compréhension, le contrôle et la prévision du comportement du système lorsqu'il devient complexe, grâce à l'existence d'un grand

nombre de holons de bas niveaux. C'est au fait que ces agrégations sont engendrées par une auto-organisation et non par une assignation statique que PROSA doit sa flexibilité. Pour le cas des HR, cette propriété permet la reconfiguration automatique d'un système en fonction des ressources disponibles ; pour les HP, elle permet la réconception des produits, tandis que pour les HO, elle permet l'assignation d'ordres existants à des ordres plus prioritaires.

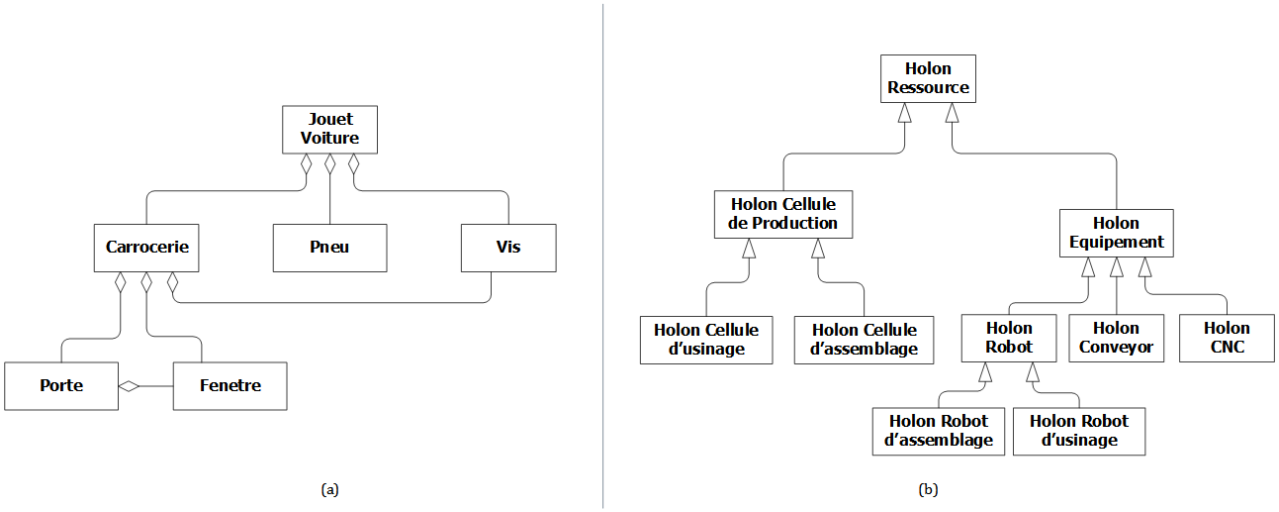


Figure 11 Exemple de : (a) Agrégation parmi des *Holons Produit*, (b) Spécialisation des *Holons Ressource*

La spécialisation divise les holons en fonction de leurs caractéristiques. Par exemple, la distinction entre les différents types de holons est déjà une spécialisation, cependant elle peut être encore très abstraite pour la représentation du système. Grâce à la spécialisation, il est possible de différencier les holons d'un même type pour qu'ils aient des objectifs différents. Pour atteindre cette variante de comportement, il faut différentes approches d'implémentation. C'est pourquoi PROSA a la capacité de représenter en détail le fonctionnement de toutes les parties d'un système.

L'architecture HCBA

Cette architecture, «*Holonic Component Based Architecture*», est proposée par (Chirn and McFarlane, 2000) et se base sur une fusion des principes du développement basé composants, en anglais « *Component based Development* » (CbD), et le concept de HMS. L'objectif d'une telle fusion est le développement d'une architecture hautement décentralisée, construite à partir de plusieurs composants modulaires autonomes, coopératifs et intelligents, capables de gérer les changements de manière rapide, en portant une attention spécifique à la reconfigurabilité du système. D'une part, c'est avec la perspective de CbD que la partie informatique est développée à la recherche de la création de composants informatiques réutilisables et reconfigurables du point de vue de l'architecture et pas seulement des modules de software individuels. D'autre part, c'est avec la perspective HMS qu'on identifie les attributs de tels modules dans une architecture dynamique. Ainsi, ces composants modulaires, autonomes, coopératifs et intelligents sont capables de gérer les changements de manière rapide.

Les composants

Comme pour le concept général d'un système holonique, HCBA est composée de deux types de composants dans le système de production, les composants ressource et les composants produits. Toutes les responsabilités du système de production sont attribués à ces deux composants, et assurent la production du produit avec une certaine succession de traitements, un certain niveau de qualité, tout cela

pour une certaine échéance. Voici une brève explication des caractéristiques des composants Produits et Ressources :

- *Composant Ressource* : Composé d'une partie physique et une partie informatique de commande, sa partie physique est dédiée à l'application des opérations, tandis que la partie commande s'occupe de l'exécution des opérations dans les ressources, la prise de décisions concernant les ressources et de la communication avec les autres composants pour la négociation. Ce sont les ressources qui sont en charge de l'ordonnancement des opérations en recherche de l'optimisation de leur utilisation.
- *Composant Produit* : Composé aussi d'une partie physique et une partie informatique, sa partie physique peut représenter des matériaux, des parties du produit, des palettes, etc. D'autre part, la partie informatique est en charge du programme de production, y compris le contrôle de routage, le contrôle du processus, la prise de décisions et les informations de production. Il s'agit d'un lien «un à un » entre ces deux parties. Dans la partie informatique, on peut aussi identifier des agents virtuels avec des rôles spécifiques. Tout composant produit est composé d'un *Coordinateur du Produit* qui crée à son tour des *Agents WIP* « *Work in Process* ». Tous les deux sont en charge de la conclusion d'un ordre, mais à des niveaux différents. Lorsque le coordinateur produit se met en charge du suivi de la production d'un lot, ce sont les agents WIP qui se mettent en charge du suivi de la production d'une pièce individuelle. Donc, l'agent WIP est celui qui fait les négociations nécessaires avec la communauté des ressources pour définir le traitement de la pièce dans le parc de ressources. Ces négociations sont faites avec un objectif défini par le coordinateur du produit.

Les relations inter-composants

La Figure 12 montre le partage d'informations parmi les composants de l'architecture. Dans HCBA, on peut identifier trois grandes étapes d'interaction, qui sont : *l'intégration statique*, *l'intégration dynamique* et le diagnostic de défauts.

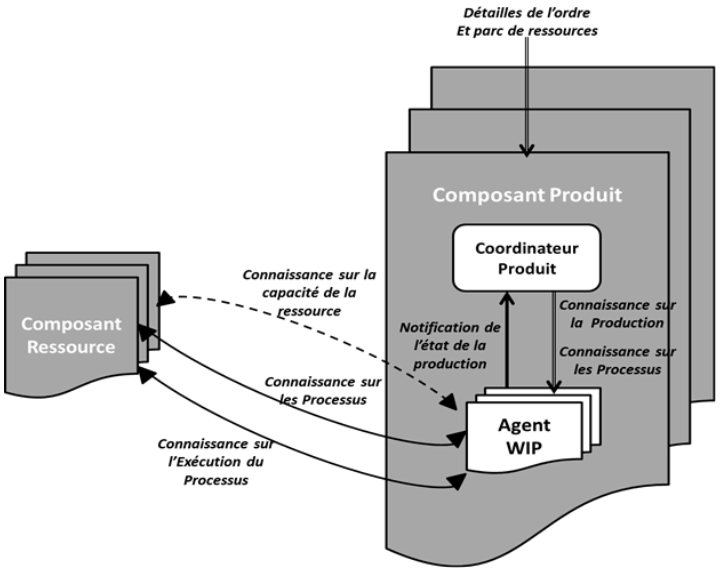


Figure 12 Partage d'informations parmi les composants en HCBA

La première étape consiste, lors de l'initialisation du système, à l'intégration de tous ses composants. Cette intégration, appelée *intégration statique*, consiste à former un parc des ressources disponibles dans le système, voire une chaine de production, qui sont au service des composants Produits pour qu'ils

puissent subir les traitements correspondants. Pendant cette étape, il n'y a pas encore d'interaction directe entre les composants, elle se fait au travers d'une base de données.

Une fois que le parc de ressources est formé, on passe à l'*intégration dynamique*. Cette étape est déclenchée par l'introduction des composants Produits. Quand ceux-ci sont introduits, ils créent des agents WIP qui, à leur tour, commencent le processus de « négociation » avec les ressources. Ces négociations engendrent l'association entre composants Produits et Ressources, selon l'état temporel du système, d'où le nom de dynamique. Dès que le dernier composant Produit a conclu son traitement et qu'il n'y a plus d'ordres, le système revient dans l'état d'intégration statique, car il n'existe plus d'interaction entre les composants.

En regardant la Figure 13 , on observe la séquence des activités des composants lors d'une Exécution-Coopération. La première étape est la création d'un ordre. En réponse à cet ordre, le système de contrôle crée un coordinateur du produit, qui à son tour crée des Agents WIP pour la production de chaque unité de l'ordre. Chaque agent WIP commence les négociations avec les ressources existantes dans le parc. Or, cette dernière étape peut se produire de différentes façons :

- *Il existe une seule ressource par opération :* les négociations ne sont alors que des demandes de disponibilité de la ressource. Ce processus est appelé *Routage Simple.*
- *Il y a plusieurs ressources offrant le même service :* Alors, l'agent WIP envoie des demandes à toutes ces ressources dans sa base de données de manière directe, il attend les offres des ressources et puis il sélectionne celui qui lui convient le plus et confirme le contrat.
- *Il n'a pas de connaissance sur les ressources :* Dans ce cas, l'agent WIP diffuse des demandes en forme d'appel à service. Les ressources répondent à leur tour s'ils ont la capacité de répondre à cet appel avec une offre. L'agent WIP attend de telles offres pendant un certain temps, puis il choisit ce qui lui convient le plus et ferme le contrat s'il est encore valide.

À la fin de la négociation, l'agent WIP obtient une *Table d'exécution* contenant l'information sur la séquence des opérations que le produit doit subir, l'équipement (ressource) à utiliser pour chaque opération et la durée de leur exécution. Or, cette table est issue des négociations effectuées à un moment spécifique avec des conditions temporelles du système. Par conséquent, les agents WIP créés par un même coordinateur vont obtenir différentes tables d'exécution.

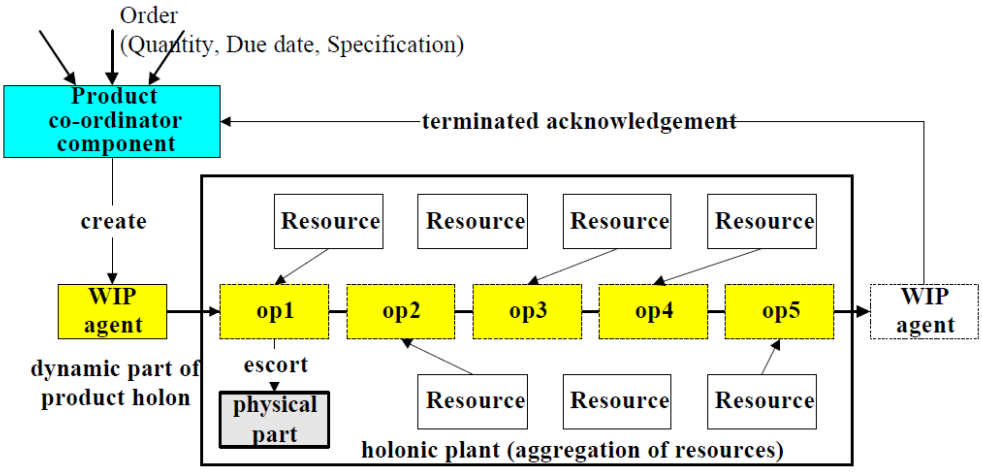


Figure 13 Interaction entre composants ressources et produits (Chirn and McFarlane, 2000)

Concernant la troisième étape, le *diagnostic de défauts* dans HCBA a lieu à travers une coopération entre Ressources et Produits, que l'on appelle *Diagnostic Coopératif*. Il consiste en ce que tout composant réalise un autodiagnostic, et en cas de défaillance, en informe les autres composants avec qui il possède une

liaison. La Figure 14 peut être utilisée pour représenter les activités de diagnostic coopératif. Pour illustrer cette étape, deux scénarios sont présentés :

Scénario 1: Une ressource subit une défaillance interne, qui est détectée par l'opération d'autodiagnostic. Elle le rapporte immédiatement aux Produits liés à elle et agit en conséquence pour sa réparation, par une action autocorrective ou par la demande d'intervention humaine. Le produit, à son tour, reconstruit alors son plan de production en considérant l'influence de ce défaut.

Scénario 2: Un Produit détecte un défaut en lui-même. Le Produit notifie alors le défaut à la ressource traitant le Produit pour agir en concordance et également à la prochaine ressource. La prochaine ressource, en fonction du défaut, changera, dans la mesure du possible, ses méthodes pour appliquer un traitement au produit en considérant le défaut ou essaiera de réparer le défaut s'il possède la capacité de le faire. Grâce à cette coopération, la capacité de diagnostic et de détection de défauts est augmentée sans augmenter le nombre de capteurs dans les composants.

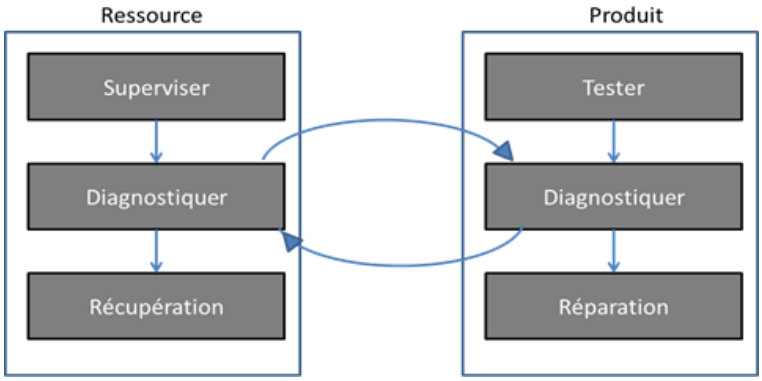


Figure 14 Diagnostic coopératif entre ressources et produits (Chirn and McFarlane, 2000)

L'architecture ADACOR

ADACOR (ADAptive holonic COntrol aRchitecture) est une architecture de référence holonique pour systèmes de production distribués proposée par (Leitão and Restivo, 2006). L'architecture a une structure de contrôle aussi décentralisée que possible mais considère dans le même temps de la centralisation dans l'objectif de tendre vers l'optimisation globale du système. Elle est basée sur un ensemble d'entités indépendantes et coopératives représentées par des holons. Chaque holon est la représentation d'un composant du processus de production qui peut soit être une ressource physique (Machines-outils à commandes numériques, robots, automates programmables industriels, palettes, etc.), ou une entité logique (commandes, produits, etc.). Les holons sont différenciés par les classes suivantes : les *holons Produits* (*ProdH*), les *holons Tâches* (*TH*), les *holons Opérations* (*OpH*) et les *holons Superviseurs* (*SupH*), interconnectés suivant le schéma de la Figure 15 .

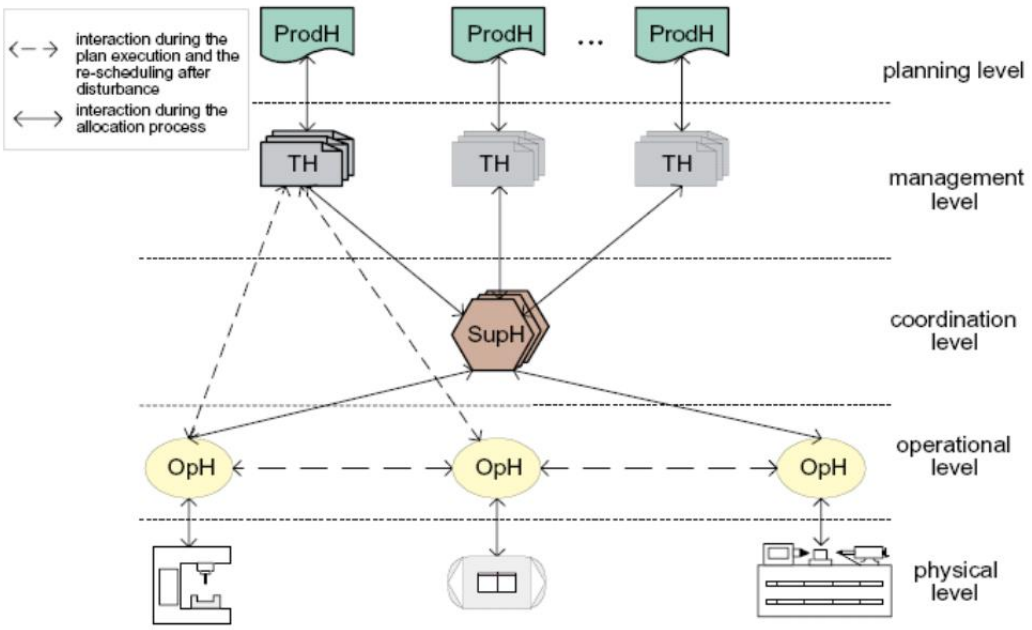


Figure 15 Répartition des holons dans l'architecture ADACOR (Leitão and Restivo, 2006)

Les *SupH* sont inspirés des systèmes biologiques et présentent des caractéristiques différentes de celles du *HS* de PROSA. Ils représentent l'un des aspects les plus innovants de l'architecture et ont été créés de manière à introduire de la coordination et une optimisation globale au sein des solutions de contrôle décentralisées. Un *SupH* peut coordonner des *OpH* ou d'autres *SupH*. Un *OpH* peut consister en une série d'*OpH* ou *SupH*, ce qui permet la construction d'holarchies fractales.

Les holons d'ADACOR sont basés sur le principe de « plug & produce ». Un nouvel élément peut donc être ajouté au système sans avoir besoin de réinitialiser ou reprogrammer le système de contrôle, ce qui amène plus de flexibilité pour la reconfiguration du système. Quand un holon apparait, il s'annonce et offre ses services aux autres holons. Lorsqu'il disparait, les holons restants doivent être capables de trouver des solutions alternatives de manière à poursuivre la production.

En plus de définir les rôles, comportements et interactions de ces holons, ADACOR a également introduit la possibilité de faire évoluer dynamiquement l'holarchie entre un état appelé stable (pendant lequel le système est sensé fonctionner de manière optimale) et un état transitoire (adopté en cas de perturbation de l'état stable). Une évolution de ce mécanisme a été présenté dans (Barbosa et al., 2015) sous le nom de ADACOR². L'objectif est de ne pas limiter l'architecture aux deux états cités précédemment, mais permettre au système d'évoluer dynamiquement au travers de configurations découvertes online. Le reste de l'architecture reste toutefois globalement constant par rapport à ADACOR.

1.4 LES ARCHITECTURES ORIENTEES-SERVICES (SOA) : FLEXIBILITE AU NIVEAU DU PROCESSUS

1.4.1 Introduction à SoA

Dans le domaine de l'informatique, notamment dans les applications Web, il existe un type d'architecture bénéficiant d'un grand succès pour la conception de systèmes agiles, autonomes et reconfigurables. Ce paradigme est celui des Architectures SoA : « *Service-Oriented Architecture* » (Komoda, 2006). Ce style d'architecture se caractérise par un contrôle décentralisé qui décompose les processus

computationnels d'une application en taches indépendantes pour ensuite les distribuer parmi les différentes ressources de traitement disponibles, bénéficiant ainsi du parallélisme et d'une distribution de la charge computationnelle dans le traitement du processus complet. Ces tâches ont alors d'attribuées une identification et une description propres ainsi que certains autres éléments pour décrire leur portée et sont modélisés comme des *Services*. Le paradigme de « Service- Orientation » représente une série de principes pour la conception et le déploiement de systèmes interopérables pour la création d'un contexte où des systèmes considérés hétérogènes puissent exposer leurs fonctionnalités comme un ensemble de services granulaires et autonomes (ou de faible couplage) à travers des interfaces standards bien définies, lesquelles peuvent être utilisées au sein de multiples applications.

Selon la définition de (Grönroos, 2000), un service représente une tache (ou une série de taches) de nature quasi-intangible, qui a normalement lieu au sein des interactions entre consommateur et fournisseur, offert par le fournisseur comme une solution à la problématique du consommateur. Un tel concept de services permet de profiter des bénéfices d'*interopérabilité* et de *réutilisation* dans le domaine de processus au travers de la création de blocs fonctionnels réutilisables, appelés services, représentant des opérations via une interface opaque qui encapsule toute information relative à leur implémentation. (Molina et al, 2005).

Les technologies de *Web-Services* ont été le véhicule d'implémentation le plus utilisé pour l'application des principes de ce paradigme. (Jammes and Smit, 2005) proposent l'utilisation de cette technologie comme interface d'accès universel pour résoudre le problème d'interopérabilité entre des dispositifs provenant de différents constructeurs et utilisant différentes technologies y compris les technologies héritées. Cependant, le paradigme n'est pas spécifiquement orienté Web. Concept plutôt issu d'une stratégie marketing, l'orientation-services en soi ne fait aucune référence a quelque technologie ou mécanisme d'implémentation spécifique que ce soit (Perrey and Lycett, 2003). Le concept de SoA, et donc celui de services, est plutôt basé sur les abstractions utilisées dans des systèmes distribues de grande taille. Etant donné qu'il est relativement compliqué de trouver une définition généralisable et unifiée sur laquelle on puisse s'appuyer pour définir les développements SoA, une approche populaire a été de définir les premiers services dans un domaine abstrait (Perrey and Lycett, 2003). De ce fait, plusieurs définitions suggèrent que le concept de service est juste un autre terme pour interface, mais une interface qui opèrerait sous des accords (contrats) définis par une sémantique spécifique, fournie sous la forme d'une *ontologie*.

1.4.2 Principes de l'orientation-services

Pour réussir la conception de systèmes bénéficiant de la réutilisation de code et de l'interopérabilité, le paradigme de l'orientation-Service suggère l'application de principes de base que sont : l'*encapsulation*, l'*abstraction*, la *granularité*, *l'autonomie*, la *réutilisation*, la *compossibilité*, le *découplage* et la *définition de contrats*. Voici quelques détails sur chacun de ces principes :

- *Abstraction et Encapsulation* : Toute implémentation de la fonctionnalité d'un service est invisible au consommateur du service. Seules les données relatives aux résultats du service tels les détails relatifs au contrat avec le fournisseur de service sont à sa disposition. Toute la complexité d'un service est donc encapsulée dans une interface opaque et abstraite.
- *Granularité :* Un service peut représenter une fonctionnalité complexe à haut niveau tout autant qu'une fonctionnalité dite atomique de très bas niveau. Le choix de la granularité doit se faire en vue de l'application : une fine granularité permet plus de flexibilité de composition mais ajoute une complexité dans la gestion et augmente le nombre des messages, tandis

qu'une baisse de la granularité limite la flexibilité de composition et rend des fonctionnalités plus élémentaires inaccessibles.

- A*utonomie* : Un service possède tout contrôle sur la fonctionnalité qu'il encapsule et son implémentation est indépendante d'autres services du même niveau. Le succès d'un service n'est pas lié au succès des autres services.
- *Découplage*: Un service sert d'interface entre un consommateur de service et son fournisseur. Le premier n'a aucune connaissance sur le deuxième et se sert donc de l'information décrite dans le contrat du service. La fonctionnalité d'un service peut être fournie par tout composant implémentant la même interface (service) et peut être remplacé sans affecter le consommateur du service.
- *Compossibilité :* Les opérations atomiques comme les opérations complexes sont encapsulées et exposées comme des services dans l'architecture. Des Services atomiques peuvent composer des processus pour que ceux-ci, à leur tour, soient aussi représentés par des services d'une plus grosse granularité. Ces *services composés* sont présentés comme des services atomiques du fait que toute connaissance sur leur implémentation serait en violation du principe de découplage.
- *Réutilisation :* Du fait qu'un service est une abstraction d'une fonctionnalité et qu'il est autonome de tout autre service, un service peut être réutilisé dans différents processus et dans différents contextes, limité par la portée de l'ontologie définissant sa sémantique.

En plus de ces services, le paradigme SoA propose aussi, pour la conception d'une plateforme indépendante de toute technologie, l'implémentation de mécanismes de *découverte* des services, laquelle doit être ouverte et accessible aux consommateurs de services. Pour l'échange d'informations dans une telle plateforme interopérable, le paradigme suggère une *communication basée sur les messages,* où les informations sont bien structurées et où la communication se fait de façon *asynchrone* en vue de garder le découplage des composants. Le format des messages doit être standardisé, libre de toute technologie d'implémentation, comme peut l'être par exemple le format XML.

1.4.3 Couches abstraites de SoA

Pour mieux expliquer les exigences de conception d'une architecture orientée-services, (Linthicum, 2012) propose avec une perspective holistique que SoA soit formé de trois couches abstraites de capabilité, qui sont exposées ; la *Couche d'Exposition*, la *Couche de Composition*, et la *Couche de Consommation*. La couche d'exposition se concentre sur l'exposition de ressources existantes comme une collection de services standardisés dans un réseau de composants interconnectés. Dans cette couche, il s'agit de déterminer comment présenter les services au système en termes de richesse de description et identification. La couche de composition se concentre sur les stratégies et les protocoles pour la composition des flux complexes de processus à travers différentes combinaisons des services individuels délivrant le résultat souhaité. La flexibilité atteinte à ce niveau sera déterminée en fonction des choix faits dans la couche d'exposition en relation à la granularité avec laquelle les services ont été conçus. Finalement, la couche de consommation se concentre à déterminer la façon d'invoquer des services comme services simples ou complexes.

Dans le contexte de la production, cette couche se concentre plutôt sur la définition de stratégies pour l'exécution des plans de production établis dans la couche de compossibilité. Les aspects qui concernent ce niveau sont la coordination des ressources et des produits, et c'est justement dans cette étape que les stratégies pour la réactivité seront définies afin de répondre rapidement face à la présence d'une perturbation.

1.4.4 Perspectives de services

La notion de perspective est un aspect important dans la conception de systèmes orientés-services. Ceci vient du fait que la perception de la valeur d'un service, décrit en termes de fonctionnalité, est une question relative à la perspective du consommateur, tandis que l'implémentation du dit service est à la responsabilité du fournisseur. La valeur de la rétribution de son exécution prend alors une perspective toute différente, Figure 16 (a). Le service lui-même représente la frontière entre ces deux perspectives de la valeur (Pahl and Zhu, 2006).

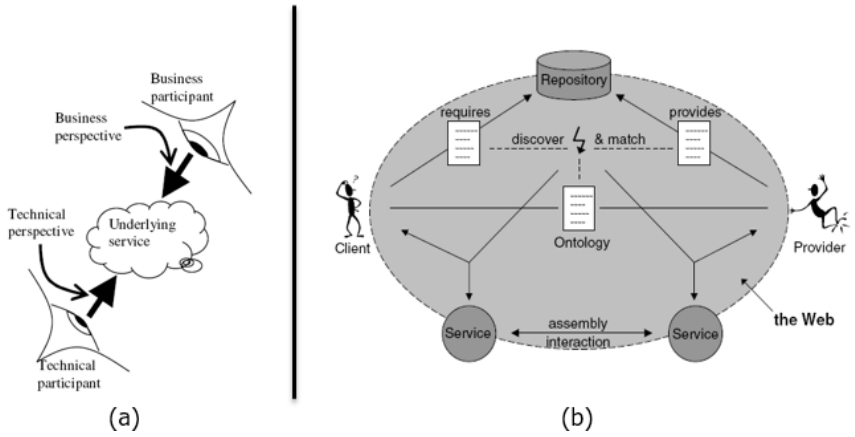


Figure 16 (a) Perspectives du consommateur et fournisseur, (b) Plateforme orienté-services (Pahl and Zhu, 2006)

Ainsi donc, le consommateur voit un service comme une unité de transaction (qui peut être de transformation d'un élément physique ou informatique), qui définit ce qui est fait mais pas comment cela est fait. D'un autre côté, le fournisseur voit le service comme une unité de fonctionnalité abstraite du contexte pour lesquels des technologies pour leur implémentation sont déjà existantes. La Figure 16 (b) illustre la relation entre les différents éléments de l'architecture et les perspectives.

Dans le domaine des Services Web-Sémantiques, une branche de la technologie des Services Web, l'ontologie OWL-S (Martin et al., 2004), définit trois perspectives de services, chaque perspective contenant des informations essentielles à propos du service qui s'adapte aux besoins de chaque couche abstraite. Ce sont les *Service Profile,* le *Service Model* et le *Service Grounding,* illustrés dans la Figure 17 :

Service Profile *:* Le modèle de cette perspective contient toutes les informations relatives aux agents chercheurs de services pour déterminer si un service porte la fonctionnalité nécessaire pour combler ses propres besoins. Son modèle comprend des informations sur les conditions d'exécution ainsi que des descriptions des effets apportés après son exécution. Elle sert comme carte de présentation d'un service pour répondre à la question « Qu'est-ce que le service peut faire pour des clients candidats ? »

Service Model *:* Son but principal est de répondre à la question « Comment un service est-il utilisé/composé ?» en fournissant des informations sur le contenu sémantique des requêtes de service et les conditions de son exécution. Pour les services composés, cette perspective indique aux clients comment construire un processus à partir de l'invocation de ses étapes intermédiaires.

Service Grounding : cette perspective indique aux consommateurs de services comment interagir avec le service et comment l'invoquer. Le service grounding fournisse les informations nécessaires pour l'invocation du service, ainsi que le contenu sémantique de l'interaction. Son objectif est de répondre à la question « Comment interagir avec le Service ? ».

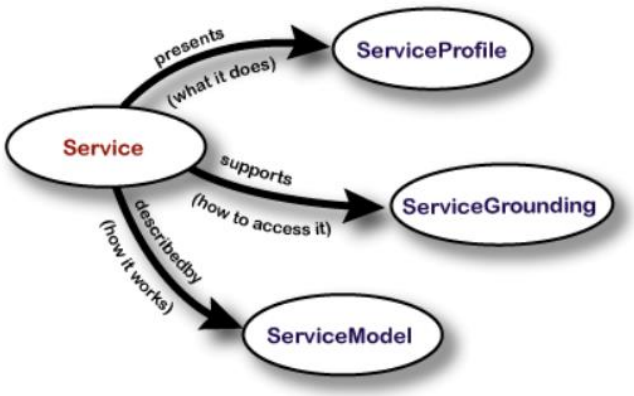


Figure 17 OWL-S ; perspectives d'un service (Martin et al., 2004)

1.4.5 SoA dans l'industrie

Même si le paradigme des SoA a été conçu pour son implémentation dans des applications informatiques au niveau Entreprise, les principes et pratiques de conception d'une architecture distribuée sont aussi applicables au domaine de la production. La mise en œuvre des principes SoA au niveau atelier (au niveau du Manufacturing Exécution System – MES – plus précisément) peut potentiellement apporter les mêmes avantages que SoA sur les applications au niveau Entreprise.

Cette vision est confirmée par l'existence de différents travaux portant sur l'application des principes SoA au niveau industriel. Par exemple, on trouve les travaux de (Jammes and Smit, 2005) décrivant les opportunités et les défis que présentent l'intégration de SoA pour le développement des « Next Generation Embedded Devices » (Dispositifs Embarqués de Prochaine Génération) et ainsi créer des réseaux des dispositifs interopérables. Ceci est proposé au travers d'une « Device Profile for Web Services (DPWS) » (Profil de Dispositif pour les Services Web) qui est une collection de protocoles standardisés pour la découverte des services, leur adressage, la notification des évènements, messagerie, et autres fonctionnalités permettant l'exposition des services d'un dispositif dans un réseau avec d'autres dispositifs offrant aussi des services. De même, (Komoda, 2006) envisage l'application de SoA à différents niveaux d'un système de production. À un haut niveau, i.e. au niveau administratif de la production, de la gestion des matières premières, de l'inventaire, des ordres de fabrication, de l'ordonnancement de la production et du suivi des activités de production général, on peut appliquer les principes de SoA d'une manière directe du fait que les caractéristiques au niveau Enterprise y sont similaires. À un niveau intermédiaire, soit au niveau des contrôleurs des cellules de fabrication, l'application des SoA continue à être prometteuse mais les contraintes d'exécution temps-réel de ces ressources posent toujours des difficultés. En effet, passer des ordres de niveau supérieur au niveau de l'atelier présente des contraintes temporelles difficiles. L'application de SoA pour ce niveau nécessite des temps de réponse estimés de l'ordre de 10 ms, de supporter une orientation à évènements et de supporter des applications asynchrones et parallèles. Au niveau le plus bas, celui du terrain, (Komoda, 2006) suggère que l'application de SoA n'est pas applicable due à la forte exigence d'une exécution strictement temps-réel. Cependant, des travaux comme ceux de (Legat and Vogel-Heuser, 2015) suggèrent le contraire, s'appuyant sur leur expertise à la fois académique et industrielle au niveau des APIs pour intégrer à ces équipements un dérivé le plus fidèle possible de la notion de service. Le DPWS (Guinard et al., 2010) est une technologie multidisciplinaire pour la communication inter-machines basée sur les services Web. DPWS emploie des mécanismes d'échanges de messages similaires aux *Web Services Architectures (WSA)* avec des restrictions quant à la complexité et à la taille des messages. Il permet des échanges de messages sécurisés, une découverte dynamique des services et une description des dispositifs basée sur *WS-*

Discovery, *WS-MetadataExchange* et *WS-Transfer* (Jammes and Smit, 2005). De plus, il fournit un mécanisme de publication-souscription basé sur *WS-Eventing* (Moritz et al., 2009).

De plus, des nombreux projets Européens se concentrent sur l'application de SoA dans l'industrie. Parmi ces projets, on trouve notamment les projets SIRENA, SoDa et SOCRADES. Le Projet SIRENA (Bohn et al., 2006), acronyme pour « Service Infrastructure for Real Time Embedded Network Applications » (Infrastructure de Services pour Applications de Réseaux des Dispositifs Embarqués Temps-Réel) est un consortium composé de 15 partenaires de trois pays européens. Ce projet a posé le travail de base pour le développement d'une SoA industrielle, Figure 18 Sa voie de recherche vise la création d'un framework orienté-services pour la spécification et le développement d'applications distribuées dans divers environnements d'application. De tels environnements ne sont pas limités aux applications industrielles, mais peuvent s'étendre à d'autres environnements où l'on trouve des dispositifs embarqués tels que la domotique, les transports et les télécoms. Ce projet a prouvé que l'approche SoA peut être appliquée avec succès aux micro-dispositifs à bas coût. Les projets SoDa (Gschwind et al., 2006) et SOCRADES (De Souza et al., 2008) sont issus de ce dernier. SoDa a pour objectif le développement d'un écosystème pour concevoir, construire et lancer des applications basées sur des dispositifs qui mettent en valeur les apports du projet SIRENA. SOCRADES (Infrastructure orientée-services et inter-couches pour dispositifs embarqués intelligents et distribués) est un projet formé par 15 partenaires appartenant à 6 pays Européens dirigé par Schneider Electric. Son objectif principal était de créer une infrastructure pour la production orientée-services, dans laquelle des dispositifs intelligents peuvent interagir avec d'autres composants orientés-services. Ces interactions peuvent exister au niveau application autant qu'au niveau dispositif. Le projet a aussi démontré que les systèmes hérités (usuellement des systèmes avec d'anciennes technologies) peuvent être intégrés dans un écosystème orienté-services au travers d'un médiateur ou d'une passerelle. Ce projet vise à construire une réelle coopération dans la production grâce à des outils d'orchestration ou de chorégraphie de services.

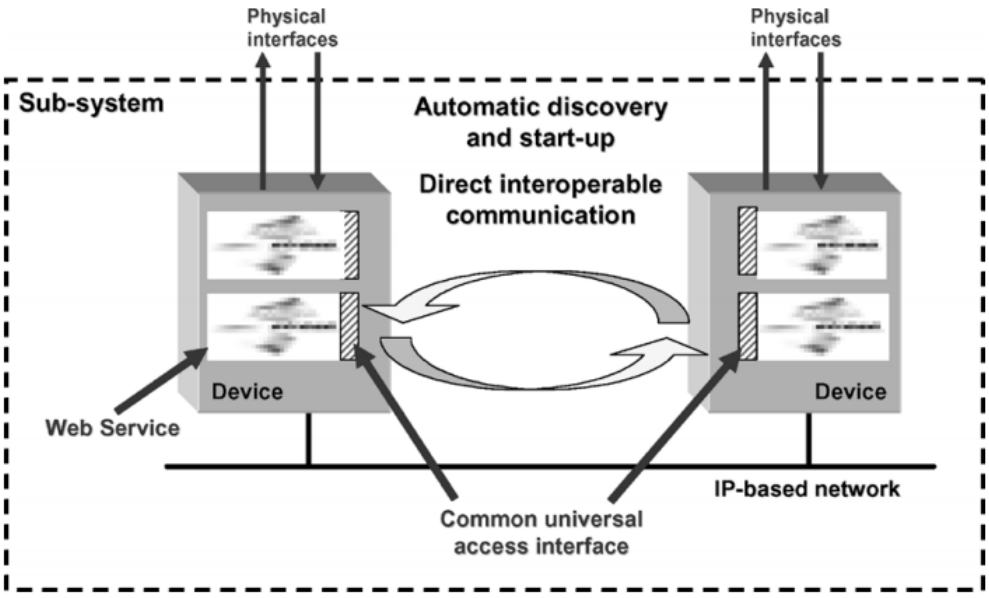


Figure 18 Perspective du projet SIRENA par (Jammes and Smit, 2005)

Un aspect vers lequel tous les travaux convergent est la liste des avantages que SoA peut amener dans une application industrielle. Parmi ces avantages on trouve :

- *Facilité dans l'intégration, réutilisation et reconfiguration d'équipement :* L'abstraction entre l'implémentation et l'interface des services permet l'implémentation des services dans n'importe quel dispositif ou plateforme informatique. Des services existants peuvent être facilement intégrés avec d'autres services pour créer des services composés de plus haut niveau. Usuellement, près d'un tiers du coût d'une production se situe dans les opérations d'installation et de mise au point (Jammes et al., 2007), ce qui peut être considérablement réduit en adoptant les principes de répétitivité et réutilisation.
- *Composition Récursive et Agrégation de Services :* Les services dans un système de production peuvent être composés par plusieurs services atomiques plus granulaires en services de plus haut niveau. Ces services atomiques sont des services réutilisables qui se répètent parmi différents processus dans un domaine ou une application.
- *Découplage entre les aspects physiques et logiques :* Un tel découpage permet l'intégration de systèmes hérités du fait que SoA est centrée processus et non centrée technologie. L'abstraction de l'implémentation des services et de leur interface permet ainsi d'intégrer les systèmes hérités au travers d'un médiateur ou d'une passerelle pour exposer la fonctionnalité de ces systèmes.
- *Développement simplifié de nouvelles applications :* La facilité d'intégration entraine aussi une facilité de reconfiguration. Quand les besoins de production changent (soit la modification de certaines caractéristiques d'un produit ou le développement d'un nouveau produit), les services interopérables peuvent être recombinés pour répondre aux nouveaux besoins avec un minimum effort. De même, les implémentations de services peuvent être modifiées sans affecter les consommateurs de services, qui ne voient que l'interface des services et non leurs méthodes d'implémentation. Cet avantage se traduit aussi en un plus court temps entre la conception et la production.

1.5 CONCLUSION

Ce premier chapitre avait pour objectif de présenter les voies possibles issues de la littérature d'amélioration de la flexibilité des architectures de contrôle des systèmes de production. De cette étude en sont ressortis deux concepts que sont les systèmes de production holoniques issus du monde de la productique et les architectures orientées-services issues du monde de l'informatique.

Après avoir présenté les origines et les fondements du paradigme holonique, nous avons détaillé les architectures holoniques de référence existantes dans la littérature et la manière dont sont agencées les relations entre les différentes entités de contrôle du système. Nous avons ensuite présenté les fondements des architectures orientées-services et pointé quelques travaux utilisant ces architectures dans un contexte de production industrielle.

Le reste de ce document est construit autour de l'idée du couplage entre les deux concepts afin de créer un système de production holonique orienté-services permettant au système de contrôle d'avoir le plus de flexibilité possible. Le chapitre suivant est très centré sur la notion de services, et notamment comment ce concept s'adapte aux caractéristiques des systèmes de production.

Chapitre 2

Adaptation des Services au Contexte de la Production

Dans ce chapitre, l'on considère le concept de service, présent dans les architectures orientées-services et originalement conçu pour des applications distribuées, notamment des applications Web, et l'on propose un modèle informationnel pour son adaptation aux applications industrielles de production. Ce modèle informationnel est adapté d'une telle manière que l'on peut construire, et ainsi décrire, les capacités d'un atelier de fabrication ainsi que les activités de production. Cette adaptation est aussi faite pour la description du caractère fractal des architectures holoniques. Les services deviennent alors, au sein du système de pilotage, les éléments principaux de description des négociations et d'échange.

Dans un premier temps, ce chapitre décrit le besoin d'adapter le concept de Service pour satisfaire les intérêts des systèmes de production, notamment pour la spécification des Produits avec pour objectif leur production et la spécification des Ressources pour leur description et intégration au système de pilotage. La relation entre le développement des Familles de Produit et des Familles de Processus est décrite afin de relever les avantages d'une conception modulaire et d'une réutilisation concordant avec le concept de services. Dans un deuxième temps, les modèles de service de production (MService pour « Manufacturing Service ») est présenté, ainsi que des modèles de processus orientés-services basés sur le modèle adapté de service. Finalement, comme il sera présenté, les services de production existent avec différentes perspectives dans les systèmes de pilotage, selon l'acteur qui l'utilise et son intention. Ces perspectives seront utilisées notamment pour la découverte, la composition et l'exécution des MServices dans une architecture Holonique de Production orientée-Services, comme il sera présenté dans le chapitre suivant.

2.1 BESOIN D'ADAPTATION POUR LA PRODUCTION

Comme mentionné dans le chapitre précédent, le concept de service représente de grands avantages pour la conception d'architectures logicielles. Elle est une architecture centrée processus où le principal but est la décomposition des activités en sous-processus pour leur distribution parmi les ressources disponibles, au sein d'un réseau interconnecté. De plus, comme détaillé dans la section 1.4.5, ce concept a déjà été largement analysé, et évalué comme une solution attractive et intéressante pour des applications industrielles. Plusieurs projets ont porté sur l'intégration de leur principes et pratiques avec l'objectif de créer des environnements industriels agiles et reconfigurables, avec une intégration rapide et une interopérabilité des différents dispositifs.

Cependant, le concept de services comme les technologies pour leur implémentation ont été conçus originalement pour les applications au niveau Entreprise et pour les applications Web, si bien que la

nature de ces systèmes ressemble à ceux des systèmes de production, les intérêts dans ces deux domaines variant légèrement. Dans le domaine de l'informatique, par exemple, le but principal est la décomposition et distribution de la charge computationnelle de traitement parmi des ressources interconnectées dans un réseau distribué. Dans le domaine de la production, ceci reste encore vrai, mais on doit ajouter d'autres intérêts. Relativement à la définition du système, les systèmes de production ont besoin d'outils (modèles) pour décrire les opérations capables de générer les transformations physiques et informationnelles sur l'environnement et sur les produits. Ces mêmes outils doivent être adaptés à la description des capacités de ressources présentes dans l'atelier de fabrication. Relativement au fonctionnement et à la performance du système, les systèmes de production doivent chercher un équilibre entre la flexibilité et leur performance et termes de planification et ordonnancement de la production. Ajouté à cela, les outils développés pour répondre à ces intérêts doivent être conçus pour leur adaptation aux nouveaux paradigmes de contrôle, tels que le paradigme holonique spécifiquement ciblé dans cette thèse.

Les travaux présentés dans le chapitre précédent ont apporté de grandes avancées dans l'intégration du concept de services dans des environnements industriels, notamment autour des standards et technologies pour leur implémentation. Toutefois, jusqu'à présent, on ne trouve pas encore de travaux portant sur la définition formelle des éléments décrivent et composent un service dans le contexte de la production. Il existe donc le besoin d'un modèle de service de production contenant les éléments décrivant la nature des transformations qu'un service représente. En concordance avec les intérêts des systèmes de production, l'objectif est de créer un modèle de service à partir duquel on puisse former des spécifications de produit complètes, ainsi que des spécifications de compétences des ressources pour leur découverte et leur prise en compte dans les processus de planification et d'ordonnancement.

Ainsi, la création d'un modèle de service de production permettra, dans une architecture holonique, de représenter les processus de fabrication de façon modulaire, sous la forme d'un service de production ayant une identification et description propres. Grâce à la décomposition et l'encapsulation des processus de production, les HMSs pourront profiter d'une flexibilité augmentée au niveau processus. Les processus de production pourront ainsi, grâce à la modularité et à la répétitivité des services, être orchestrés (séquencés) de différentes manières, respectant les relations entre modules, elles-mêmes représentées par un autre outil modélisant la structure d'un processus.

2.1.1 Adaptation pour la spécification des produits

Les besoins que doit satisfaire une méthodologie pour la spécification des produits dans le cadre d'une production basée sur les services peuvent être vus comme la somme des plusieurs conditions, caractérisant l'environnement de production comme décrit dans l'introduction de cette thèse. D'un côté, il y a les exigences d'un marché en évolution constante avec des cycles de vie raccourcis et une personnalisation accrue des produits. D'une autre côté, il y a la nature fractale des produits : un produit peut être la collection de sous-produits, qui peuvent être eux-mêmes des collections d'autres sous-produits plus granulaires et ainsi de suite. Enfin, le paradigme des architectures orientées services encourage la répétitivité et la réutilisation des opérations.

Tous ces facteurs exigent alors d'un outil pour la spécification d'un produit les caractéristiques suivantes :

- La spécification des produits doit être capable de représenter la relation fractale entre le produit et ses sous-produits.
- L'outil de modélisation des produits doit permettre la customisation au niveau scalaire et modulaire pour capturer avec un même modèle toutes les variations possibles dues à la spécification du consommateur.

- Pour répondre aux principes de conception des architectures orientées service, l'outil de modélisation des produits doit être composé de façon modulaire. La conception de ces modules (les services de production) doit être orientée vers la répétitivité des modules pour exploiter leur réutilisation dans les spécifications d'autres produits similaires.

De plus, pour augmenter l'intégration et l'interopérabilité entre atelier et applications, on cherche la définition de spécification de produit qui soit indépendante de toute information liée à l'outil de production. Ainsi, une spécification produit doit pouvoir être lancée dans n'importe quel atelier comprenant les capacités exigées par la spécification produit.

2.1.2 Adaptation pour la spécification des ressources

Dû au fait que l'on parle de l'intégration des principes des architectures orientées-services dans une architecture de pilotage holonique, il est important que le modèle décrivant un service de production s'adapte aussi à représenter et encapsuler le caractère fractal que l'on retrouve dans la structure des ressources. Les ressources les plus agrégées offrent des services de haut niveau. Ces services sont, tout comme les ressources qui les offrent, composés par d'autres services d'un niveau plus granulaire et ainsi de suite jusqu'à que l'on trouve des ressources indivisibles offrant des services atomiques.

Le principe d'intégration dicté par les SoA demande aussi un modèle de service qui soit capable de représenter la flexibilité inhérente des ressources. En effet, les avancées dans le domaine de la robotique et de la mécatronique amènent à construire des ateliers de plus en plus polyvalents et reprogrammables, capables d'une vaste gamme de capacités de transformation. Dans le même temps le coût élevé de ces nouvelles technologies ne va pas forcément de pair avec la rénovation des ateliers, ce qui implique une intégration de ces nouvelles technologies plus facile si l'on se base sur les technologies héritées. Pour répondre à ce principe d'intégration et d'interopérabilité, le modèle de service de production doit être capable de modéliser la flexibilité induite par les technologies émergentes. Le modèle doit aussi motiver l'exploitation de cette flexibilité au travers de la programmation de méthodes hautement paramétrables pour leur réutilisation dans d'autres applications. En outre, du fait de la présence des différentes technologies et des intérêts de la production comme la qualité ou l'utilisation optimisée de l'outil de production, le modèle de service de production doit aussi comporter des éléments pour l'évaluation du couple service-ressource. Dû à la présence des différentes technologies dans l'atelier de production, un même service peut en effet être offert avec différents attributs en fonction de la ressource qui l'offre ou des méthodes utilisées pour l'offrir.

Les opérations dans un système de production ont une nature toute différente par rapport aux services du domaine de l'informatique. Les opérations et activités au sein de l'outil de production comportent des transformations de caractéristiques de produits et de l'environnement ainsi que l'agrégation d'autres caractéristiques ou composants. Le modèle adapté doit ainsi comporter les éléments informationnels pour représenter de manière qualitative et quantitative la manière avec laquelle un service modifie l'environnement dans lequel il agit.

Ainsi, avec un modèle de service adapté à la production, les capacités des ressources pourront être déterminées par la collection des services de production qu'elles offrent, tout service portant des attributs pour leur évaluation et différentiation avec les autres ressources. La flexibilité des ressources pourra être exploitée grâce au paramétrage des services. Cette flexibilité pourrait aussi bénéficier de la réutilisation des méthodes et former des nouveaux services de plus haut niveau. Pour créer de telles nouvelles fonctionnalités, il suffira donc de modéliser les relations entre les services granulaires, sans avoir besoin d'intervenir directement dans l'atelier. La programmation pourra se faire offline et être mise en ligne avec un simple téléchargement de la déclaration du nouveau service.

2.2 RELATION AVEC LES FAMILLES DE PRODUITS

2.2.1 Les familles de produits

Les entreprises manufacturières, dans leur intention de rester compétitives dans un marché avec une demande croissante de produits personnalisables, tentent d'élargir leur offre des produits. La « *mass customatization* » (personnalisation de masse) a pour objectif d'augmenter la variété dans l'offre des produits d'une entreprise pour ainsi augmenter leur attractivité et par conséquent, leurs ventes. Cependant, la gestion d'une offre avec une grande variabilité ne vient pas sans poser un certain nombre d'inconvénients. En effet, cette variété entraine une augmentation de la complexité interne du système de production, ce qui implique une augmentation dans le coût de production (Child et al., 1991). Une solution de plus en plus adoptée par les entreprises pour faire face à cet antagonisme entre l'augmentation de l'offre des produits et la réduction de la complexité interne du system est le *Développement de Familles de Produits*. Cette approche est bien reconnue comme une solution permettant aux entreprises manufacturières d'atteindre une économie d'échelle sans perdre de vue la satisfaction des besoins du consommateur (Martinez et al., 2000). Le développement des familles de produits est basé sur l'exploitation des similarités inhérentes que l'on peut trouver parmi les différentes variantes des produits. Une famille de produits fait référence donc à un ensemble de produits individuels partageant un ensemble de caractéristiques structurelles, tandis qu'ils sont différenciés par d'autres caractéristiques spécifiques (Meyer and Lehnerd, 1997). Les produits individuels, dérivables d'une même famille de produits, sont alors considérés comme des membres, ou encore des instances d'une famille de produits, si l'on se réfère à un terme issu de la programmation orientée objets.

Outre la similarité, la *modularité* est une autre caractéristique importante dans le développement des familles de produits. La modularité, dans une architecture de produits, fait référence au degré avec lequel on peut décomposer la structure d'un produit en blocs individuels et répétables dans la structure des autres membres de la famille, ou encore dans d'autres familles de produits. Ces modules représentent un regroupement de composants physiques autant que conceptuels, e.g. le groupe des sous-produits constituant un produit, le groupe des caractéristiques issues des transformations sur un produit de base, etc. Un aspect clé de la modularité est l'interaction et l'association entre les modules structurels. Cela implique une concordance entre les interfaces permettant un degré d'interaction parmi les différents modules d'une structure. Ce degré d'interaction parmi les modules, pour créer des structures modulaires, est alors défini par les efforts fait pour la standardisation des interfaces comme pour la définition des règles de configuration des structures modulaires. Ainsi donc, une famille de produits peut se construire à partir d'une collection des modules représentant les caractéristiques des produits et une liste des relations parmi ces modules décrivant leur structure, de manière analogue à la construction des structures à partir de blocs Lego®.

En plus de la similarité et de la modularité, une autre propriété exploitée par le développement des familles de produits est la *répétitivité*. En effet, les bénéfices amenés par la conception des familles de produits, e.g. la réduction de la complexité interne, sont dus au profit (ou à son encouragement dans la conception) de la répétitivité des modules. La conception des produits est simplifiée par le regroupement de modules déjà validés pouvant ainsi créer un grand spectre des variantes en variant le choix et la configuration des modules. De même, cette répétitivité se traduit en une réutilisation des composants, donc des processus, ce qui réduit les coûts liés à la gestion de la complexité et ceux liés à la conception et programmation des processus.

Dans la littérature, on peut trouver deux types de plateformes de personnalisation pour le développement de familles de produits : les plateformes scalaires et les plateformes configurables (aussi connues comme

modulaires) (Ulrich and Eppinger, 1995), (Du et al., 2001). Ces plateformes décrivent la façon avec laquelle on différencie les membres d'une famille. Dans les plateformes de caractère scalaire, la différenciation se fait en faisant varier les valeurs d'un ou plusieurs Paramètres de Conception (PC) quantifiables (Simpson et al., 2001), par exemple le diamètre de perçage dans une plaque ou la capacité de mémoire RAM dans un ordinateur. Dans les plateformes de personnalisation modulaires, la différenciation parmi les variantes des produits se fait en modifiant la configuration des modules existants par l'agrégation, soustraction et /ou substitution d'un ou plusieurs modules structurels (Meyer et al., 1997), e.g. le choix du clavier dans un ordinateur portable ou le choix d'ajouter ou non une webcam. Dans la suite de cette thèse, lorsque l'on parlera d'une plateforme de personnalisation, on fera référence à une plateforme combinant ces deux degrés de personnalisation, i.e. au niveau scalaire et au niveau modulaire. Dans ce scénario, un module structurel dans une structure de produit peut être personnalisé quantitativement en éditant ses paramètres de conception tandis que sa présence dans la structure peut être modifiée en fonction des interfaces entre les modules et le choix du client.

En résumé, l'approche basée sur les familles de produits, pour répondre au besoin de personnalisation, entraine de nombreux avantages en termes de conception et du côté économique. Plus précisément, on peut lister du point de vue de la personnalisation et de la conception de produits :

- *Une personnalisation des produits facile* : Grâce à la création des modèles de familles de produits, les consommateurs peuvent créer leurs propres variantes de produits en modifiant les paramètres des modules proposés ainsi que leur configuration en fonction des règles définies par les modèles ;
- *Une réduction du temps de conception de nouveaux produits* : Grâce à la répétitivité et à la réutilisation, de nouveaux produits peuvent être conçus sur la base de modules déjà validés, réduisant ainsi les risques de conception. Ainsi, le temps de mise sur le marché (Time to Market, TTM) est réduit considérablement, incrémentant ainsi l'agilité des entreprises.

Et du point de vue économique, on peut lister :

- *Une réduction de l'inventaire de composants* : Grâce à la réutilisation et la répétitivité des composants parmi les produits de l'offre, l'inventaire des composants peut être considérablement réduit en quantité et en variété, réduisant ainsi les coûts de gestion (Fisher et al., 1999) ;
- *Une réduction des types de machines de production* : Une réduction dans la variété des composants implique aussi une possible réduction de la diversité des ressources dans l'atelier.

Au niveau de la production, la plupart des bénéfices potentiels du développement des familles de produits sont néanmoins à chercher du côté du développement similaire de familles de processus. La section suivante présente cette déclinaison.

2.2.2 Les familles de processus

Sur la base des idées exposées dans (Martinez et al., 2000; Schierholt, 1999), la similarité trouvée dans la conception structurelle d'une famille de produits se traduit usuellement par une similarité dans les processus de production réalisant les produits membres d'une famille. Autrement dit, les modules des caractéristiques d'un produit se traduisent usuellement dans des modules de processus représentant des opérations, processus et/ou séquences dans le domaine de la production.

À partir de telles idées émerge le concept de *Familles de Processus* qui, de la même manière que les familles de produits, sont caractérisées par les attributs de similarité, modularité et répétitivité mais dans le domaine des processus (Jiao et al., 2007; Meyer and Lehnerd, 1997; Simpson et al., 2001). Une famille de processus est alors une collection de modules représentant des opérations de production qui répondent à

la réalisation des modules d'une famille de produits. Ainsi donc, dans le modèle informationnel d'une famille de produits existe la description formelle, par un outil de modélisation, d'une structure de produit (spécification de produit) et d'une structure de processus (spécification de processus) (Jiao et al., 2007).

Il se trouve que ce concept de familles de processus ne s'éloigne pas des concepts décrits dans les architectures orientées-service. Ces modules de production peuvent être encapsulés et représentés par une interface donnant une propre identification et description et peuvent être offerts comme des services de production. Grâce à ceci, les modules de production peuvent être standardisés et être rapidement disponibles pour former des processus de production des différents produits ou de différentes familles. Parmi les avantages amenés par ce concept dans le domaine de la production, on trouve :

- *Réduction de la variété des processus* : Comme dans le domaine de la conception de produits, travailler avec un nombre réduit des procédés facilite les efforts d'entretien et ainsi réduit les coûts liés ;
- *Réduction des efforts de reprogrammation* : Le transfert de la répétitivité des caractéristiques physiques entraine une répétitivité des opérations et méthodes au niveau atelier. La programmation et la conception des processus réalisant des nouveaux produits est alors aussi réduit grâce à la réutilisation de code (à l'identique de ce que les principes de SoA dictent) ;
- *Réduction de la complexité pour accueillir les effets de la personnalisation au niveau atelier* : Dû au fait que la personnalisation scalaire se limite à un module à la fois, les effets d'une personnalisation se réduisent aussi à un seul service de production. Dû à leur nature indépendante, il devient plus facile de programmer des méthodes capables de reproduire la gamme de variantes que le service modélise. Au niveau modulaire, la personnalisation se fait par l'ajout et le séquencement des services. Il suffit d'éditer les relations entre les modules dans l'outil utilisé pour leur modélisation ;
- *Réduction du temps et coût de la mise en production des nouveaux produits :* Grâce à la réutilisation des caractéristiques de produits et de méthodes déjà validées, le temps investi dans la programmation et la conception de nouveaux procédés est considérablement réduit. Dans certains cas, il se réduit au temps nécessaire pour de regrouper les services correspondants dans une spécification de processus ;
- *Donner un degré de liberté aux fonctions des systèmes de pilotage :* Grâce à la modularité ajoutée aux processus de production, l'outil de planification et d'ordonnancement d'un processus de production peut explorer un autre degré de liberté. Ce degré de liberté permet au système de pilotage de créer différentes séquences de productions gouvernées par les contraintes relationnelles entre les services de production. Pour cela, il existe le besoin d'un outil modélisant les relations modulaires. Un tel outil sera proposé dans le Chapitre 4.

Ainsi donc, les principes du développement de familles de produits et de processus avec les principes des architectures orientées services ont le potentiel d'exploiter le principe de répétitivité et de réutilisation, entrainant leurs avantages dès la conception des produits jusqu'à leur production au niveau atelier. L'implémentation de ces deux paradigmes augmente le degré d'agilité dans les étapes de conception et laissent le terrain au système de pilotage pour exploiter la flexibilité ajoutée à la spécification des processus modulaires pour ainsi créer différentes orchestrations de processus (Chapitre 4).

2.3 LES SERVICES DE PRODUCTION (*MSERVICES*)

2.3.1 Définition

Dans un système de production, notamment au niveau atelier, le concept de services peut s'étendre à la description et l'encapsulation des différents types d'activités. De ce fait, on peut modéliser différents types des services de production. Dans le système de pilotage d'une chaine de production ou même d'une chaine logistique, on peut alors trouver des services de transport, de maintenance, de production, d'approvisionnement, ou encore de répertoire, etc., comme illustré sur la Figure 19 .

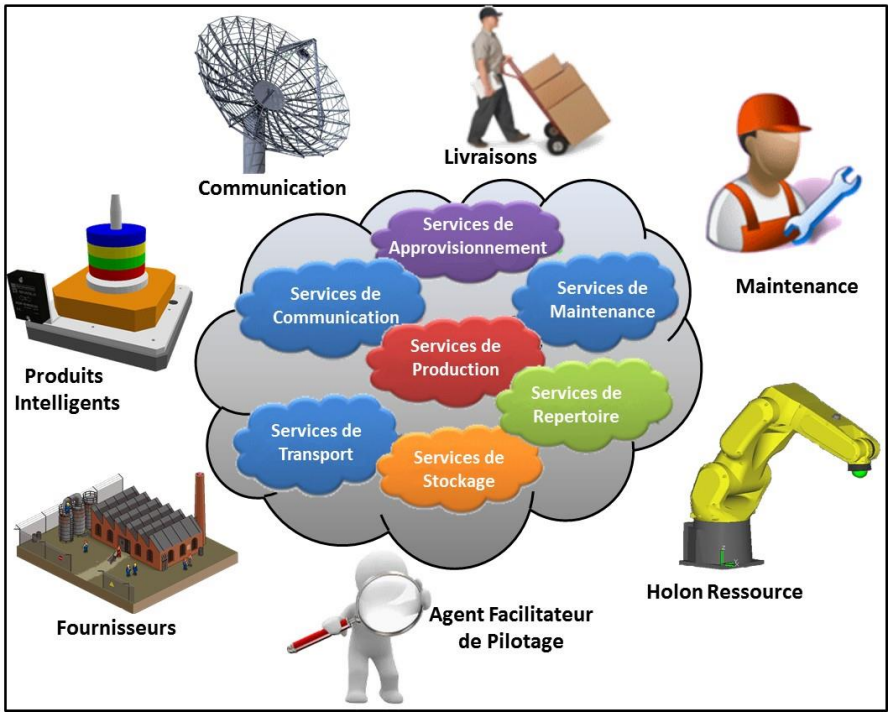


Figure 19 Les Services pour modéliser différents activités dans la production

Les *services de transport* décrivent les opérations disponibles pour déplacer des produits ou sous-produits d'un point à un autre du système de production, ou bien pour représenter un transport inter-systèmes. *Les services de maintenance* peuvent décrire des activités préventives ou correctives pour l'entretien des ressources. Ces services peuvent représenter des opérations automatisées ou bien des interventions humaines, et peuvent être inscrits dans le plan d'utilisation d'une ressource pour les prendre en compte lors de l'élaboration des stratégies de planification de la production, afin de pouvoir s'exécuter conjointement sans conflit. Les *services d'approvisionnement* représentent des activités nécessaires à la gestion du stock et inventaire des produits et sous-produits. Les ressources détectant des niveaux bas de stock peuvent faire appel à un service d'approvisionnement. Les *services de répertoire*, plus liés à la plateforme de pilotage, représentent des opérations liées à la découverte et au registre des ressources et produits, ainsi que d'autres informations relatives au système. Ce sont des services permettant d'accéder à des informations globales, potentiellement hors de portée du demandeur. Les *services de communication,* aussi liées à la plateforme de pilotage, sont des opérations pour le transport et routage de messages parmi les acteurs du système. Les *services d'outillage* sont des services passifs utilisés par le système de pilotage pour réserver l'utilisation de ressources passives et partagées, comme des outillages ou des espaces de travail. Les *services de port* sont aussi des services passifs, sont utilisés de manière conjointe avec les services de transport et représentent l'utilisation d'un port du système. Un port est un point physique d'entrée ou de sortie de produit lié à une ressource. Enfin, il y a les *services de production (MServices).* Ceux-ci représentent

des activités plus liées à l'atelier de production. Les services de production peuvent être divisés en plusieurs catégories selon une taxonomie, comme illustré dans la Figure 20 . Un service de production représente une ou plusieurs opérations de transformation des produits ou de l'environnement de production (e.g. usinage, traitement de surface, assemblage, contrôle, emballage, etc. ou bien une combinaison de plusieurs).

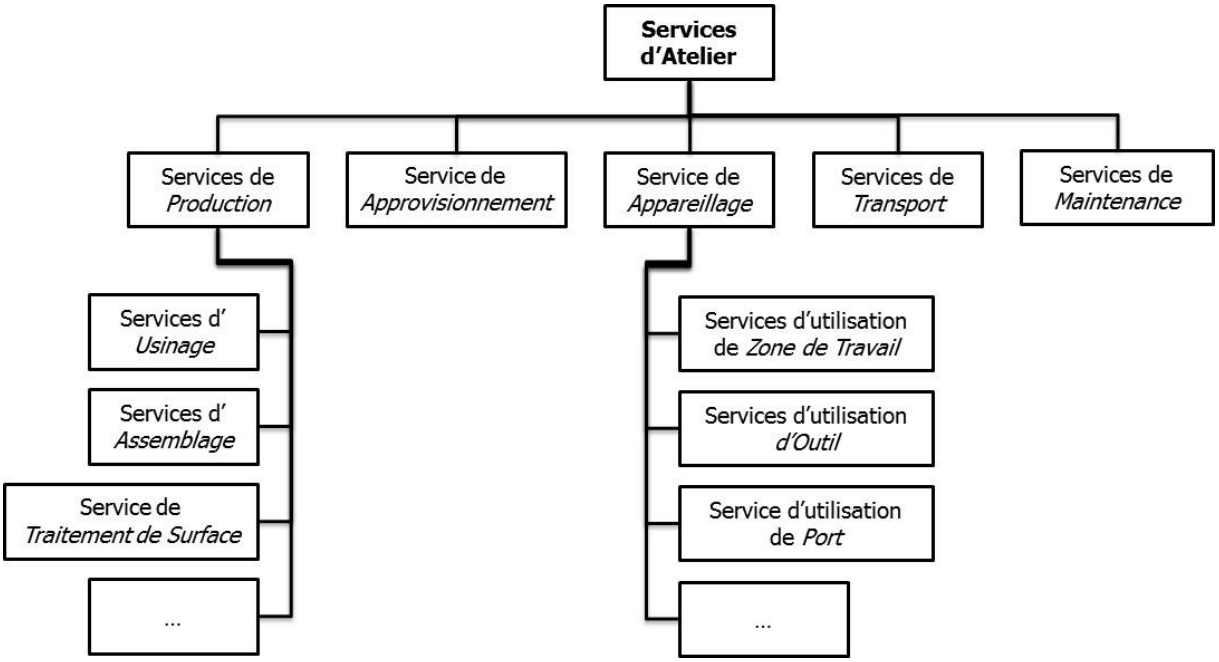


Figure 20 Exemple de taxonomie de MServices

Dans ce chapitre, et dans le reste de cette thèse, on se concentre sur la modélisation de services de production. Dans les sections suivantes, un modèle informationnel est proposé pour décrire les éléments qui composent un service de production, lequel sera la base des spécifications produit comme des spécifications de ressources et deviendra l'élément principal des négociations dans le système de pilotage.

2.3.2 Modèle conceptuel des MServices

En rappelant la définition de (Grönroos, 2000), un service peut représenter une activité singulière ou bien une série d'activités d'une nature plus ou moins tangible qui peut faire partie des interactions entre un client et un fournisseur comme une solution pour atteindre les objectifs du client. Ainsi donc, comme décrit dans la section précédente, on peut définir un *Service de Production* (MService) comme l'encapsulation et la représentation d'une opération ou d'une série d'opérations qui ajoutent de la valeur à un produit semi-fini Un tel ajout de valeur peut prendre la forme d'une transformation du produit ou bien d'une agrégation. Un *MService* est modélisé comme une O*pération Non-Interruptible (ONI)* c'est à dire une opération dont l'exécution amène un produit à un état stable, connu et désiré sans tenir compte des méthodes utilisées dans l'opération. Après l'exécution d'un *MService,* le produit en question passe à un état apte pour le stockage et/ou le transport.

Avant de passer à la définition du modèle informationnel exploitable par le système de pilotage, il devient utile de faire une analyse conceptuelle des éléments qui vont former un *MService* et une analyse de la nature des relations entre ces éléments. La Figure 21 présente ce modèle conceptuel est modélisé en langage UML (Unified Modeling Language), langage largement utilisé en informatique pour modéliser des relations entre éléments d'un système telles que les relations d'agrégation, dépendance, spécialisation, etc.

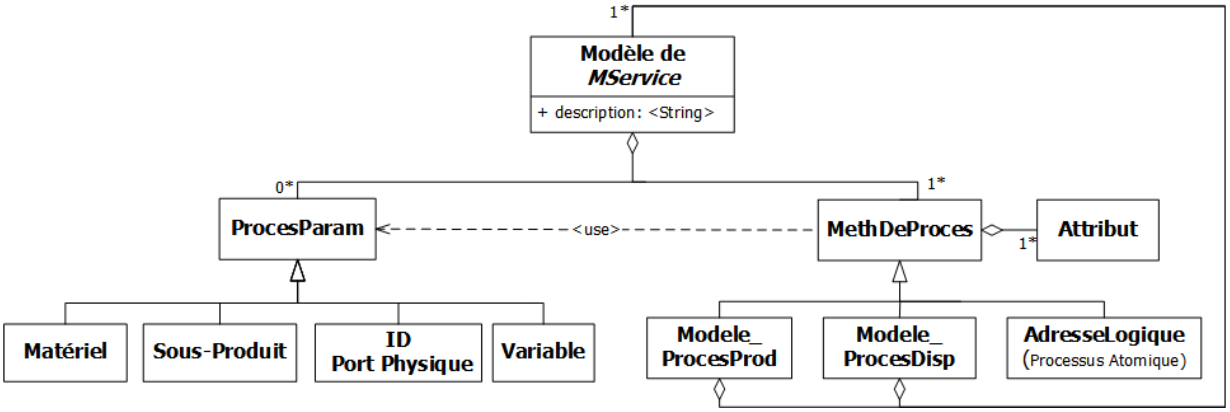


Figure 21 Modèle conceptuel d'un *MService*

Un *MService* peut être composé d'une ou plusieurs *Méthodes de Production*, par un ensemble de *Paramètres de Processus* et par un ensemble d'*Attributs* :

Description du MService : Tout *MService* a besoin d'une étiquette pour sa classification et, plus important, la description de la nature des transformations qu'il reproduit. Cette description peut aussi comporter la liste des préconditions et des effets du *MService.*

Méthode de Processus de Production (MethDeProces) *:* Elle représente le procédé utilisé pour reproduire les transformations décrites et stipulées dans la description du *MService.* Contrairement à la description des MServices, cette partie est propriété du fournisseur de services car elle est indépendante de la description du service. Comme sa cardinalité l'indique, plus d'une méthode peuvent être associées à une description de service. Trois types de méthodes ont été identifiés dans un atelier : *Processus - Produit*, *Processus -Dispositif* et *Processus Atomiques.* Les deux premiers représentent des *MServices* composés tandis que le dernier représente une opération indivisible, comme un appel à l'exécution d'un programme contenu dans un contrôleur par exemple. Ceux-ci seront présentés dans les sections suivantes.

Paramètres de Processus (ProcesParam) *:* Ceci comprend les informations nécessaires aux méthodes de production pour délimiter l'échelle des transformations à réaliser. Un paramètre peut fait référence à quatre types différents, soit une variable quantitative, une propriété qualitative, la spécification d'un matériau ou bien la spécification d'un sous-produit par exemple. Comme sa cardinalité l'indique, il peut exister autant de paramètres que nécessaires.

Attributs *:* Les attributs comprennent des informations relatives à l'évaluation d'éligibilité des ressources et des méthodes. En effet, les valeurs des attributs ne sont pas liées à la description d'un service mais aux méthodes qui les implémentent. Une méthode peut être plus ou moins performante qu'une autre selon certains critères, qui sont mis à la disposition du système de pilotage par les attributs.

Ce modèle conceptuel montre deux relations importantes : (i) La relation de dépendance entre les paramètres de processus et les méthodes et (ii) la relation d'auto-agrégation indirecte des *MServices* à travers des méthodes représentant des processus complexes non-atomiques.

Relation paramètres-méthodes

Il existe une forte dépendance entre l'ensemble des paramètres et les méthodes créées pour satisfaire l'implémentation des services. Les algorithmes des méthodes ont besoin de la spécification des paramètres pour déterminer l'échelle et/ou les caractéristiques des transformations à réaliser. Cependant,

le point le plus important de cette relation est la séparation de ces deux éléments, i.e. le découplage des paramètres et des méthodes. Cette séparation des paramètres comme partie de la description des *MServices* permet l'intégration de la personnalisation de produits au niveau le plus bas de la production, i.e. les MServices. Un *MService* peut être vu de manière analogue à une famille de processus, partageant une même description mais différenciés par les valeurs de leur paramétrage. Les efforts faits pour la conception de méthodes paramétrables permettent d'augmenter leur réutilisation dans différentes familles de produits. La granularité d'un service est importante dans la recherche de flexibilité et sa réutilisation (Den Haan, 2007).

Relation d'auto-agrégation indirecte : les *MServices Composés*

Dans le diagramme de la Figure 21 on peut observer une relation indirecte d'auto-agrégation des MServices. Cette relation dépend de la ressource que fournit le *MService* et notamment de la méthode qu'elle implémente. Avec cette auto-agrégation indirecte, un *MService* peut être composé d'autres *MServices* plus granulaires en fonction des types de méthodes utilisées. Ces services formés par une collection de *MServices* plus granulaires sont appelés *MServices Composés*. Un service peut alors offrir un même processus comme service composé et/ou service atomique. Cette information est propriété de la ressource et n'est pas présente dans la description du service ni exposé au client. Ceci facilite l'intégration éventuelle de différentes technologies dans le système de production, ce qui augmente la flexibilité de reconfiguration et d'extension du système. La fractalité du modèle des *MServices* est mise en évidence avec le concept de *MService* Composés. Cette propriété fractale entraine plusieurs avantages :

- Elle reflète la structure d'une famille de processus comme une formation des sous-processus déjà validés ;
- Elle permet l'extension des compétences du système à travers la combinaison et structuration des *MServices* déjà présents ;
- Elle s'adapte à la structure fractale des ressources dans un HMS ;
- Elle s'adapte à la structure fractale des produits avec la spécification des matières premières et sous-produits comme paramètres. Par exemple, on peut indiquer l'agrégation d'un sous-produit en tant que paramètre d'un *MService* d'assemblage.

En effet, une architecture holonique demande des modèles qui s'adaptent à son caractère fractal, où l'on peut modéliser des ressources et des produits comme un tout ou bien partie d'un tout de plus haut niveau. Ce modèle conceptuel, avec la spécification des sous-produits et la propriété d'auto-agrégation, permet cette adaptation.

2.3.3 Paramétrage des *MServices*

Dans une description de *MService,* les paramètres de processus peuvent avoir deux perspectives : *ProfParam* et *SpecParam*. Le diagramme UML de la Figure 22 montre la relation de ces perspectives.

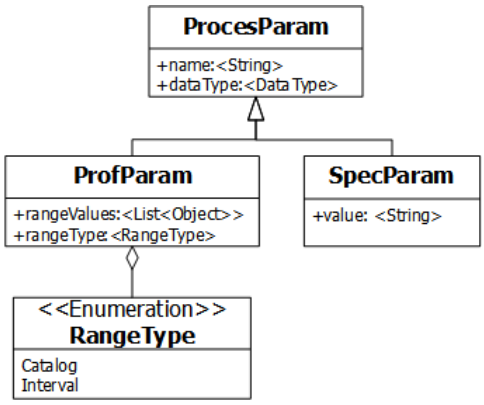


Figure 22 Perspectives des Paramètres de Processus

Comme indiqué précédemment, tout paramètre de processus comporte une identification propre et une description de ce qu'il représente, e.g. une variable, une propriété, etc. Reliée à cette description, une *Spécification de Paramètre (SpecParam)* indique la valeur des caractéristiques que le paramètre peut prendre. Son but est d'instancier un paramètre en une valeur spécifique, issue des négociations avec le client du *MService* en question.

A l'opposé, un *Profil de Paramètre* (*ProfParam*) ne sert pas à indiquer une valeur spécifique mais une plage de valeurs que le paramètre, selon sa description, peut prendre. Cette perspective est uniquement utilisée pour décrire les capacités d'un *MService* par rapport à la caractéristique en question. La plage de valeurs du profil peut être décrite sous la forme d'un *catalogue* de valeurs qualitatives ou comme un *intervalle* de valeurs quantitatives, voire d'un catalogue d'intervalles.

La réutilisation des services s'appuie sur leur paramétrage. C'est grâce au paramétrage que les *MServices* peuvent englober une grande gamme de transformations, qui peut être instanciée à travers la spécification des paramètres. La compréhension de ces perspectives est utile pour appréhender les modèles informationnels d'un *MService,* présentés section 2.5, qui sont présents dans le système de pilotage avec différentes perspectives selon leur utilisation.

2.3.4 Accessibilité des MServices

Du point de vue des clients, un *MService* peut être vu comme une boîte noire, comme une opération atomique amenant un produit dans un état stable conforme à la description du service. Les *MServices* sont de fait perçus et invoqués comme des *processus simples*. Un processus simple, comme décrit dans OWL-S, est un élément d'abstraction utilisé pour fournir une vue à un processus atomique (ou bien une vue simplifiée à un processus composé). C'est-ainsi que tous les *MServices* offerts dans le système de production sont perçus par les clients des services comme des boites noires, sans avoir besoin d'impliquer les clients dans l'exécution des services. Les ressources offrant les services ont la responsabilité de gérer et garantir l'exécution du service au nom du client. Cette perspective contribue également à l'extension du système et à l'intégration des systèmes hérités. Les *MServices* servent comme interface entre la couche de contrôle et la couche d'exécution au niveau terrain avec l'encapsulation de la fonctionnalité. Cette interface isole les informations relatives aux méthodes et technologies du système de pilotage du système de production, plutôt concerné par la coordination et l'ordonnancement des tâches.

Les technologies pour l'invocation des *MServices* est hors de la portée de cette thèse. Néanmoins, étant donné que tous les services sont considérés comme des processus simples pour le client, un seul protocole d'invocation suffit pour tous les MServices, qui sera développé au chapitre 3.

2.4 LES ONTOLOGIES DE SERVICE

Comme mentionné dans la première partie, cette thèse est basée sur l'idée que la performance et le comportement du système dépendent de la façon dans laquelle les informations sont structurées. Cette affirmation prend particulièrement du sens pour les systèmes distribués, où il existe le besoin d'une cohérence informationnelle afin de garantir l'émergence d'un comportement cohérent. La communication entre entités autonomes et distribuées utilise des messages nécessitant un contenu sémantiquement riche, qui permet une compréhension unifiée des concepts du domaine de connaissance concerné (Gangemi et al., 2001). En outre, l'intégration du système implique la capacité de tous les éléments du système à dialoguer (raisonner et comprendre) en termes de capacités pour conclure des accords.

Pour apporter une solution à ce besoin de cohérence des informations, nous proposons de spécifier explicitement les objets de l'application sous la forme d'ontologies. Ces ontologies serviront comme un vocabulaire pour la spécification des processus et des ressources, augmentant alors la clarté de définition des processus, améliorant la réutilisation des opérations conceptualisées par de telles ontologies, et dans un dernier temps contribuant à l'agilité du moteur de planification à explorer des alternatives plus performantes (Chapitre 5).En informatique, une *ontologie* représente un ensemble de concepts et de termes, donnant une signification à des champs d'information (espaces de noms) pour représenter les éléments qui appartiennent à un domaine de connaissances du monde ainsi que les relations entre ces concepts. Le but de son utilisation est de créer une base de connaissances sur laquelle on puisse raisonner à propos des objets d'un domaine. Il existe plusieurs types d'ontologies selon la portée du domaine qu'elles décrivent, Figure 23

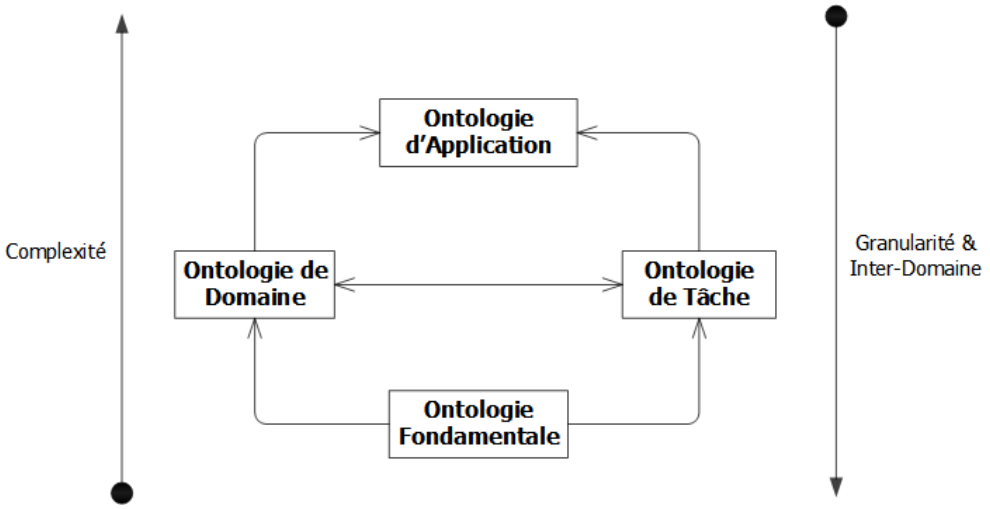


Figure 23 Topologie des ontologies

À la base de la topologie, on trouve les *Ontologies Fondamentales (OF)*. Les ontologies de ce niveau de représentation comprennent la description des concepts très généraux, que l'on peut normalement trouver dans plusieurs domaines de connaissances. Le but de ces ontologies est de favoriser l'interopérabilité entre un grand nombre d'ontologies de plus haut niveau.

Une *Ontologie de Domaine (OdD)* est une ontologie, de plus haut niveau qu'une ontologie fondamentale, décrivant une partie du monde plus spécialisée. Elle donne du sens aux termes spécifiques à un domaine.

Du fait de sa granularité, les concepts et termes d'une ontologie de domaine peuvent être définis à partir des concepts et des relations définies par plusieurs ontologies fondamentales, celles-ci étant plus granulaires.

Tout en haut de la pyramide, l'on trouve les *Ontologies d'Application (OdApp)*. L'ensemble des concepts et terminologie définis par ces ontologies est d'une grande spécialisation. Le monde de représentation se réduit aux concepts d'une application en particulier. La généralité de ses concepts est délimitée aux intérêts de l'application concernée. Un des avantages de la conception des *OdApps* est le regroupement des activités avec lesquelles on travaille dans une même description, simplifiant ainsi leur gestion.

Dans le domaine de la production, il existe plusieurs travaux proposant la définition et l'utilisation des ontologies de domaine ou fondamentales pour la description des éléments d'un système de production avec une approche sémantique (Borgo and Leitão, 2004; Delamer and Lastra, 2006). Selon (Delamer and Lastra, 2006), il existe un vide technologique dans le développement des modèles pour le « *matching* » (correspondance en anglais) des services, lesquelles nécessitent encore de la reprogrammation manuelle. Cette reprogrammation manuelle est considérée comme l'un de plus grands obstacles dans la conception de systèmes rapidement reconfigurables. Ils proposent alors la création de descriptions des services avec un contenu sémantique riche pour permettre l'emploi des systèmes de raisonnement-machine (Machine Reasoning Systems) pour mettre en œuvre un « matching » automatique des services utilisant une inférence logique plutôt que des correspondances fixes de type un-à-un. Cette approche permet d'étendre la portée de la découverte automatique des services à des cas où un « matching » direct n'existerait pas mais où les relations sémantiques inférées entre les services requis et offerts indiqueraient une correspondance. Selon (Delamer and Lastra, 2006), une ontologie, pour l'approche sémantique, nécessite :

- Une *Ontologie de Processus* : décrit une taxonomie des activités du domaine. Elle comprend un modèle de processus composé de deux parties : un modèle d'orchestration pour décrire le séquencement des activités et une partie de chorégraphie pour transformer les paramètres du service en paramètres de plus bas niveau.
- Une *Ontologie d'Equipement* : décrit des propriétés physiques des ressources, relatives à ses dimensions physiques, interfaces physiques, dimensions de processus et interfaces de processus.
- Une *Ontologie de Produit* : décrivant la structure physique des produits en termes de composants et relations entre composants. L'encapsulation des services est dérivée de la description des produits à fabriquer.

Un exemple d'ontologie de domaine est également présentée dans (Delamer and Lastra, 2006), inspirée du Standard Allemand DIN 8595 (Deutsches Institute für Normung e. V., 2003) et par (Vos et al., 2001), Figure 24

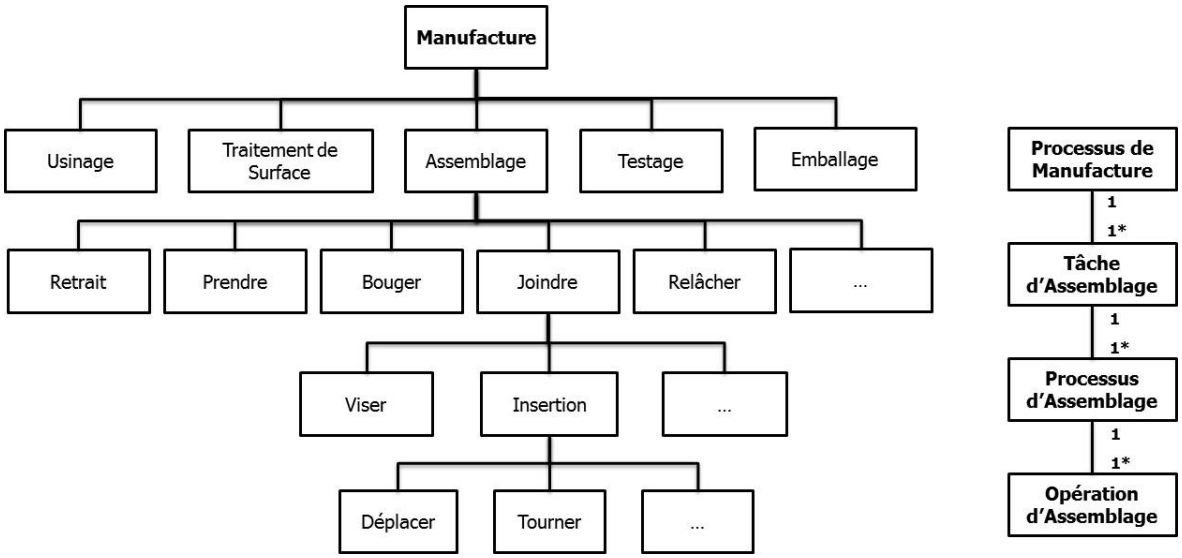


Figure 24 Taxonomie d'une ontologie d'assemblage

Cette ontologie d'assemblage peut être divisée en plusieurs librairies :

- Une libraire d'ontologie fondamentale, qui modélise les quantités et dimensions physiques, e.g. température, pression, force, vitesse, etc. ainsi que des informations géométriques et non géométriques sur les éléments d'un produit assemblé (Delamer and Lastra, 2006) ;
- Une libraire de processus pour décrire les activités ;
- Une librairie d'acteurs décrite par les services ;
- Une librairie de produits, modélisés comme une composition d'éléments portant des informations, géométriques et non-géométriques, sur la façon avec laquelle ils peuvent être assemblés pour former d'autres éléments.

Dans le domaine holonique, (Borgo and Leitão, 2004) proposent l'alignement des concepts de l'architecture de référence ADACOR avec la définition de l'ontologie fondamentale DOLCE (Descriptive Ontologie for Linguistique and Cognitive Engineering). Ils affirment le rôle crucial des ontologies fondamentales dans la création de systèmes d'application réutilisables, adaptables et transparents. Dans leur travail, ils proposent une ontologie de domaine pour les systèmes de production à partir de l'ontologie fondamentale DOLCE.

L'approche sémantique pour la définition des ontologies de domaine, par la voie des ontologies fondamentales ou non, se présente comme une solution prometteuse pour parvenir à la création de dispositifs et systèmes de production avec des grandes capacités d'auto-reconfiguration, intégration et interopérabilité. Cependant, les ontologies utilisées dans la production sont issues d'efforts non-coordonnés. De ce fait, il n'existe pas à ce jour d'ontologie formelle du domaine manufacturier ayant une large acceptation, comme il était déjà identifié dans (Borgo and Leitão, 2004). Le développement des ontologies fondamentales est un domaine de recherche relativement nouveau, et c'est seulement au sein de la dernière décennie que des systèmes bien axiomatisés et justifiés ont été proposés. De plus, comme (Pisanelli et al., 2002) le précise, le développement des systèmes d'application basés sur des ontologies est relativement exigeant, et donc seulement un nombre limité de travaux a relevé le défi. Ce constat n'est pas très étonnant du fait de la grande demande en connaissances théoriques que leur utilisation implique et du fait du grand nombre d'éléments à modéliser, comme illustré dans (Delamer and Lastra, 2006), ce qui résulte en une grande complexité de modélisation de toutes les subtilités du domaine étudié. La synchronisation des ontologies avec la réalité du domaine concerné peut là encore poser des problèmes

dans la création de spécifications représentant proprement les activités d'un système de production. Ce point prend plus d'importance dans un domaine où l'on a besoin d'informations précises et où les risques sont potentiellement importants. De plus, l'approche sémantique implique une charge de traitement computationnelle importante du fait de la richesse des informations, notamment pendant le « *matching* » des services. Ceci n'est pas négligeable dans un système où l'on vise à explorer une grande quantité d'alternatives de solutions pour la planification et l'ordonnancement de la production. Pourtant, leur application en ligne pendant cette phase est une option qui serait très adaptée aux besoins d'agilité.

Pour surmonter cet obstacle, nous proposons, dans cette thèse, la création d'ontologies d'application pour réduire la demande computationnelle de traitement, tout en gardant les attributs acquis de l'encapsulation des opérations en services. Comme mentionné auparavant, les ontologies d'application modélisent des opérations qui ne sont pas généralisables pour décrire tout un domaine, mais qui sont adaptées à une application en particulier. L'*OdApp* peut donc être vue comme une librairie des types de services relatifs à une application spécifique, i.e. des opérations relatives à la production d'une gamme de produits. Elles peuvent être construites soit par la programmation des méthodes dans le langage propriétaire des ressources en les associant à des définitions de services dans une *OdApp*, ou bien à travers des ontologies de domaine comme celui du Standard Allemand DIN 8595. En attendant que les technologies permettant l'identification des capacités et composition des services et ressources de manière automatique prennent leur essor, ce sont les ingénieurs de processus qui sont encore en charge de la création des services composés et du regroupement des ressources pour exécuter ceux-ci.

2.5 MODELES DE PERSPECTIVES DE *MSERVICES*

Comme mentionné précédemment, à la fois les SoA et les architectures holoniques ont un modèle fractal qui peut être divisé en trois couches de capacités abstraites (Linthicum, 2012). Ces couches peuvent être décrites comme:

- *Expose Layer*: détermine comment les services sont présentés au système en termes de richesse de description et d'identification;
- *Compose Layer*: détermine les stratégies et méthodes pour composer des workflows complexes en combinant des services et reproduire le résultat désiré;
- *Consume Layer*: en informatique, cette couche s'intéresse à la manière d'accéder ou invoquer un service en tant que processus simple ou complexe. En production, cette dernière couche concerne plutôt les stratégies d'exécution de *MServices* atomiques ou composés.

De manière similaire à OWL-S, le Framework est composé, selon les besoins, de différents types d'informations concernant un *MService* dans le HMS. L'information est distribuée parmi les différentes perspectives de *MService,* c'est-à-dire *Type (TypeMServ), Profil (ProfMServ)*, *Spécification (SpecMServ)* et *Implémentation (ImpMServ)*. Le diagramme UML présenté dans la Figure 21 décrit les relations entre les différentes perspectives de *MService.*

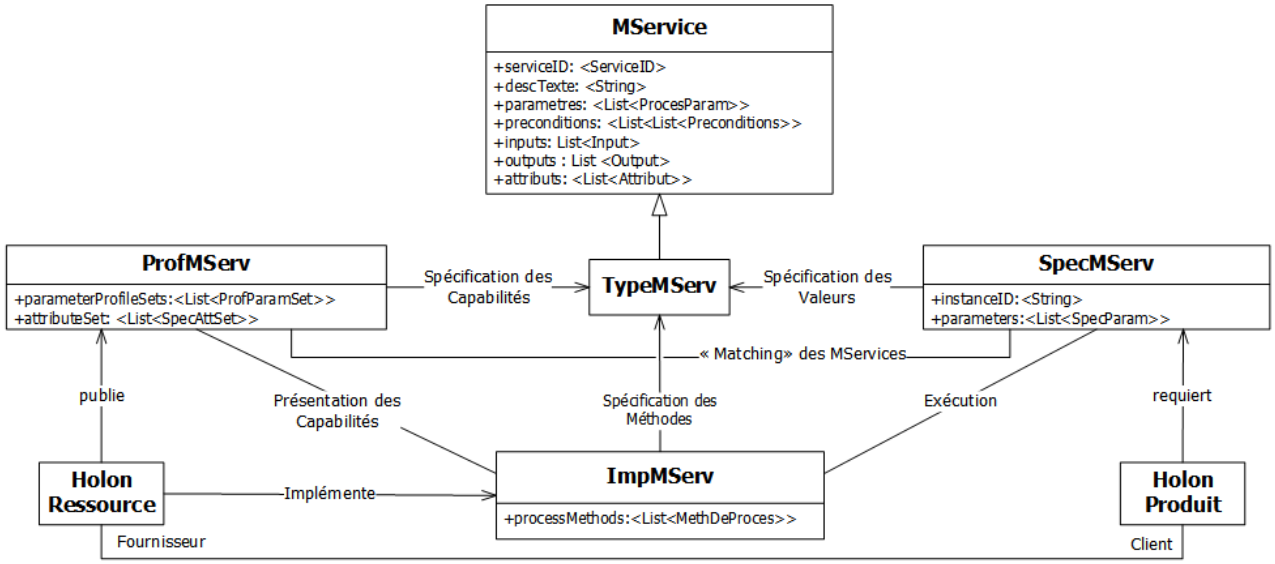


Figure 25 Perspective de service de production

En haut du diagramme, le *Modèle de MService* spécifie les éléments comprenant un service de production. La définition de ses champs, tels que le nom, la catégorie, le type de paramètres, les entrées, les sorties, etc., produit un *Type de MService* qui appartient à une application ou un domaine. Le modèle de cette perspective, *TypeMServ,* représente une classe spécifique de service à l'intérieur d'une ontologie de services qui reproduit les transformations indiquées dans sa description. Il détermine quelles sont les propriétés qui caractérisent une instance de *MService* de ce type, i.e. les propriétés qui doivent être spécifiées de manière à créer une *Spécification de MService.* Cette dernière perspective, *SpecMServ,* est donc associée à un *TypeMServ* qui fournit des informations sur la valeur de ses propriétés. Le *SpecMServ* est définie par un client, qui décrit ses besoins et utilise cette perspective pour effectuer des requêtes de services. D'un autre côté, les fournisseurs utilisent des *Profils de MService* de manière à exposer leurs capacités de transformation au reste des acteurs du système. Un *ProfMServ* fournit de l'information sur la plage de valeurs de paramètres, défini par le *TypeMServ*, qu'une ressource peut reproduire. En d'autres mots, il indique l'ensemble des *SpecMServ*s qu'une ressource peut fournir selon ses ressources internes et sa technologie. De ce fait, la découverte des associations potentielles entre fournisseurs et demandeurs de service est réalisée au travers d'une association entre *SpecMServ*et *ProfMServ,* selon un mécanisme décrit ci-après.

Enfin, la perspective *Implémentation*, *ImpMServ,* est possédée par les Holons Ressource (HR) et contient toutes les informations sur les méthodes utilisées par la ressource de manière à produire la transformation décrite par le *TypeMServ* associé. De telles méthodes sont propriétaires de la ressource car cela représente sa technologie interne, et le type de méthode peut varier selon le type de processus, comme décrit dans la prochaine section. Pour synthétiser, un modèle d'*Implémentation* exécute les opérations pour fournir la transformation requise avec une *SpecMServ,* tandis que le *ProfMServ* expose ses capacités à reproduire différents résultats, tous les deux associés au même *TypeMServ.*

2.5.1 Meta-Modèle des *MServices*

Le *Modèle de MService* contient tous les éléments nécessaires à décrire un service dans le contexte de la production. La Figure 22 illustre la composition d'un *MService.* Ce modèle peut être utilisé pour créer et ajouter des **Types de MService* à une *OdApp* ou à une *OdD* en définissant les champs des propriétés nécessaires à décrire un service de production.

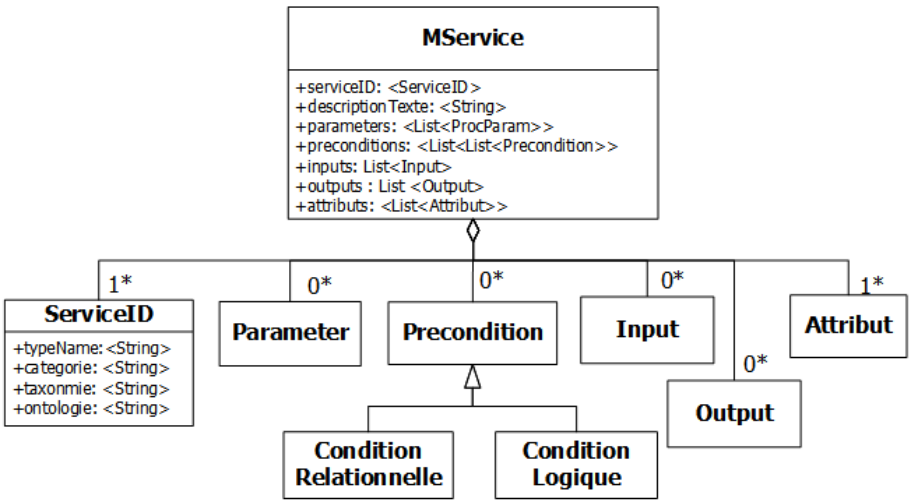


Figure 26 Modèle de *MService* (service de production)

Le premier aspect à décrire à propos d'un *MService* est son identification. Un service est identifié par un *Service ID*, définie par l'ontologie de service à laquelle il appartient, sa taxonomie et catégorie, dans une tel ontologie ainsi comme le nom du service. Le second aspect couvert est une *descriptionTexte* de ce que le service effectue dans une syntaxe compréhensible pour l'humain. C'est une description détaillée de la transformation qui peut être accomplie par le service, qui décrit les conditions initiales devant être respectées avant l'exécution et les effets apportés par le service sur le produit et l'environnement. Le troisième aspect couvre les *paramètres (ProcesParam)* d'un *MService,* Ces paramètres indiquent les informations requises pour entièrement spécifier l'intervalle d'effets qu'un *MService* peut avoir. Le quatrième aspect est lié aux *préconditions* nécessaires avant l'exécution d'un *MService.* Ces sont les conditions que le fournisseur de service et le client doivent satisfaire avant l'exécution. Lorsque les préconditions des *MServices* sont décrites pour l'humain dans le champ *descriptionTexte,* et pour la machine dans le champ *préconditions*. Le premier est lié à l'applicabilité de l'opération selon l'état du produit tandis que le second est plutôt lié à l'état du client et du fournisseur de manière à pouvoir lancer l'exécution du service.

Un autre aspect est l'information apportant des connaissances sur l'environnement qu'un fournisseur peur nécessiter pour l'exécution d'un service ainsi que l'information qu'il retourne après son exécution. Cette information est fournie par les champs *inputs* et *outputs*. Les *Inputs* diffèrent des *paramètres* dans le sens qu'ils ne sont pas liés aux propriétés du service mais aux informations nécessaires au fournisseur pour régler les bonnes conditions d'exécution. Par exemple, un modèle 3D des pinces maintenant le produit peut être donné en entrée d'un robot de peinture pour calculer les trajectoires évitant les collisions outils. Les *Outputs*, d'un autre côté, fournissent l'information pouvant compléter l'information contenue dans le champ des effets, e.g. un rapport sur la quantité de peinture utilisée par le même robot. Les *MServices* ne fournissent pas exclusivement des transformations physiques à l'environnement; ils peuvent aussi fournir des services permettant de traiter de l'information fournie en *Inputs* et retourner l'information traitée en *Outputs*.

Enfin, le dernier aspect à décrire à propos du *MService* est le critère selon lequel le service peut être évalué, i.e. ses *Attributs*. L'objectif principal du champ *Attribut* est de fournir l'information sur le critère qui sera utilisé pour évaluer la performance d'un fournisseur. Cette information peut être utilisée selon deux objectifs distincts: (i) Lors d'une requête de service, le client peut déterminer l'intervalle de valeurs amenant une performance suffisamment acceptable pour que le fournisseur soit éligible; (ii) lors d'une proposition de service, le fournisseur notifie sa performance pour la fourniture d'un service spécifique.

Des exemples d'attributs classiques peuvent être cités : Qualité de Service, temps opératoire, date de fin d'exécution, énergie consommée ou fiabilité, comme exprimés par (Rodrigues et al., 2015) dans leurs travaux sur les critères d'évaluation de services.

2.5.2 Modèle *TypeMServ*

Le modèle *TypeMServ* représente un type spécifique de service de production inclus dans une ontologie de service de domaine ou d'application. Il indique quels sont les caractéristiques et paramètres de l'opération de production qu'il représente et qui doivent être spécifiés de manière à créer les instances d'un tel *Type de MService,* i.e. une *Spécification de MService*. Le modèle de la perspective *Type*, décrit par la Figure 27 , hérite sa structure du *Modèle de MService* et définit le contenu des éléments de sa structure : *ServiceID*, *descriptionTexte*, *paramètres* et *attributs*.

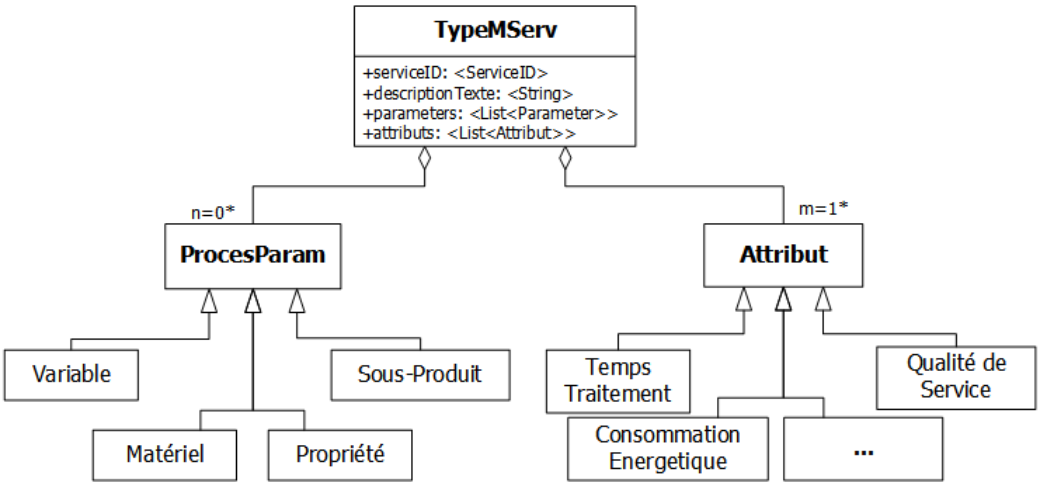


Figure 27 Modèle de Type de MService

Comme vu précédemment, les champs *serviceID* et *descriptionTexte* sont utilisés pour identifier et décrire, pour l'humain, la nature des transformations représentés par un *TypeMServ*. La perspective *Type* sert alors comme une carte de visite pour un groupe d'opérations/processus de production avec une description détaillé des résultats qu'ils comprennent. Pour un *TypeMServ* donné, une liste de *paramètres* est fournie et sert à déterminer les limites de ses transformations. Ces *paramètres* peuvent être de 4 différentes classes: (i) une *variable* décrivant une valeur discrète ou analogique du service, e.g. vitesse de rotation, tolérance, coordonnées, etc.; (ii) un *Matériel* qui sera utilisé par le service, e.g. acier, aluminium, bois, etc.; (iii) une *Propriété* de service, ou résultat, défini en termes qualitatifs, e.g. couleur = rouge, texture= polie, etc.; (iv) *Sous-Produit,* soit la spécification d'un autre produit nécessaire au service, e.g. les roues d'une petite voiture. À la fois les éléments *Matériel* et *Sous-Produit* sont ajoutés aux caractéristiques du produit. Cependant, la différence entre eux est que le premier est considéré comme un élément toujours disponible en stock tandis que le second est considéré comme produit à la demande grâce à une requête de service.

Le modèle de la perspective *Type* peut aussi indiquer des *Attributs* liés aux caractéristiques du service, qui peuvent être utilisés dans l'évaluation de l'instance de service. Les *Inputs*, *Outputs* et *Préconditions* ne sont pas spécifiés dans le *TypeMServ* car ces informations ne sont pas liées à la nature du service en lui-même, mais aux informations que le fournisseur nécessite pour déterminer si le service peut être fourni à une certaine date. Les *ProcesParams*, *Attributs*, *serviceID* et *descriptionTexte* sont spécifiques à un *Type de MService* et sont utilisés pour décrire la sorte de transformation qu'il peut fournir, ainsi que les informations nécessaires au demandeur pour compléter la description du service. Aucune information n'est fournie quant à savoir qui pourra fournir le service, comment il sera fourni ou l'échelle des transformations. En

conséquence, dans une application de production, les opérations de production se répétant constamment au cours du processus d'application peuvent être identifiées, catégorisées et décrites par des *Types de MServices*. De cette manière, une ontologie de service d'application peut être créée avec un catalogue de tous les *MServices* concernant l'application. De plus, en gardant le principe des familles de processus, les gammes de production des produits peuvent être construites à partir de ces *TypeMServs* en tant que blocs élémentaires fournissant les transformations intermédiaires au processus du produit, ce qui facilite la conception du processus et réduit d'autant le temps de mise en production.

2.5.3 Modèle *SpecMServ*

La perspective *SpecMServ* représente une instance d'un certain *TypeMServ* fournissant des informations sur les valeurs de ses paramètres et attributs. Cette perspective est définie par les demandeurs de services pour complètement définir ses besoins, en termes d'échelle de transformations..

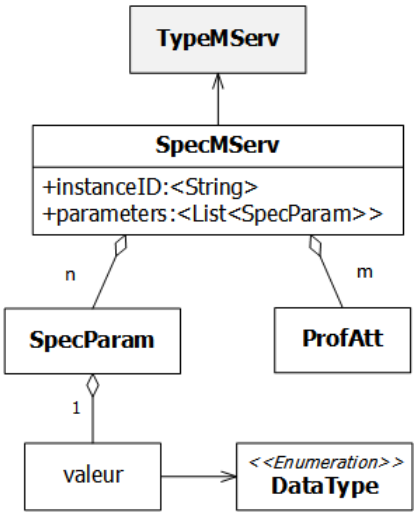


Figure 28 Modèle de Spécification de MService

Le modèle de *Spécification v* ajoute une *Spécification de Paramètre (SpecParam)* à chaque paramètre déclaré dans le *Type* associé. Ces *SpecParams* ajoutent à chaque paramètre une valeur, associée à un type de donnés indiqué par *DataType*, comme indiqué par le type de paramètre. Dans le même ordre d'idée, une liste de *Profils d'Attribut (ProfAtt)* est ajoutée pour indiquer les caractéristiques liées à la qualité du service demandé, comme expliqué précédemment. Les demandeurs de service acceptent uniquement les propositions de service ayant des attributs correspondant à ceux énoncés dans la liste de profils d'attributs.

Comme expliqué dans la section précédente, les familles de processus peuvent être construites avec une collection de *TypeMServs*. Comme dans les familles de produits, l'instanciation d'une famille de processus produit une spécification de processus d'un produit complètement déterminé, i.e. une gamme de production du produit. Les blocs élémentaires d'une telle spécification de processus sont alors les *Spécifications de MService,* chacune déterminant en détail les transformations intermédiaires du processus de production du produit. Dans un HMS, les Holons Produit (HP) sont lancés dans le système avec de telles gammes de production et annoncent leurs besoins avec des requêtes de service décrites par la perspective *SpecMServ*, et reçoivent éventuellement des propositions des HR possédant les méthodes capables de reproduire de tels services. Le HP devient alors le planificateur de la production du produit en s'engageant dans l'orchestration des possibles flux de travail *MService* composés par des différentes *SpecMServs*. Ce planning consiste en: (i) composer les séquences correctes de services; (ii) identifier les HR ayant les capacités nécessaires; (iii) négocier l'allocation des services avec les HR capables.

2.5.4 Modèle *ProfMServ*

Le *Profil* d'un *MService* est conçu pour exposer les capacités de transformation d'un fournisseur de service concernant un certain *Type*. Ces dernières, les *Types*, peuvent avoir un très large espace de transformations possibles en fonction du nombre de paramètres qu'il peut recevoir. Les ressources, avec leurs limitations technologiques, utilisent les *Profils de MService* pour indiquer le sous-espace de transformations qu'elles peuvent réaliser, en indiquant l'intervalle de valeurs de paramètres qu'elles peuvent reproduire, i.e. indiquer au travers des *ProfMServs,* l'ensemble de *Spécifications* qu'elles sont capables de reproduire selon leurs ressources et technologies internes. Deux listes sont contenues dans le *ProfMServ*, comme décrit dans la Figure 25 : une liste d'*ensembles de profils de paramètre* (*ProfParmSet*) et une liste *d'ensembles de spécifications d'attributs* (*SpecAttSet*). Ces deux listes fournissent des informations sur ce qu'une ressource peut faire. Les profils de paramètres sont regroupés en ensembles de manière à pouvoir prendre en compte les possibles couplages entre valeurs de paramètres. Par exemple, un bras robotisé offrant un service de Pick-and-Place peut être apte à réaliser l'opération dans n'importe quel angle à l'intérieur d'un espace de travail donné, mais devoir réduire ses positions angulaires accessibles dans un espace de travail plus lointain. L'angle et les coordonnées du robot, et donc du service, sont donc couplés pour cette opération. De ce fait, chaque ensemble de profils de paramètres représente un sous-espace des capacités de la ressource. De manière similaire, le *ProfMServ* contient la spécification d'un ensemble d'attributs. Etant donné que les *Attributs* sont des propriétés à destination de l'évaluation de l'exécution/performance du service, chaque ensemble d'attributs est associé à un ensemble de paramètres. Ces ensembles d'*Attributs* fournissent de l'information aux demandeurs de services sur la qualité escomptée de la fourniture de service par une ressource. En résumé, un *ProfParamSet* est un ensemble de *profils de paramètre* qui décrit dans quelle mesure une ressource peut réaliser un service et un *SpecAttSet* (ensemble de *spécifications d'attributs*) décrit à quel point il est bien réalisé.

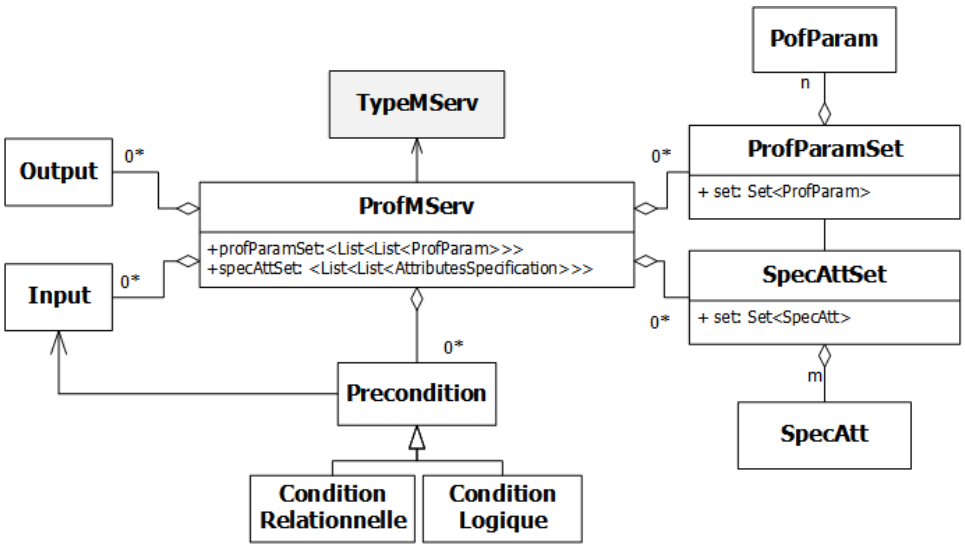


Figure 29 Modèle de Profil de MService

Les *Profils de MService* présentent aussi des informations que les fournisseurs de service ont besoin pour l'exécution d'un service. Ces informations sont données par une liste de *préconditions* et *inputs*. Les *Préconditions* sont des conditions liées à l'état de l'environnement que le demandeur et/ou le fournisseur doivent satisfaire avant l'exécution du service. Elles ne doivent cependant pas être confondues avec les conditions liées à la structure du processus, i.e. les conditions du séquencement. Celles-ci en revanche se réfèrent à des conditions externes, par exemple la présence du matériau, du produit, d'un réglage correct, d'une valeur de la température de l'outillage, etc. Les *Préconditions* sont définies comme une matrice de

préconditions logiques and/or rationnelles. Les colonnes de la matrice représentent des conditions *Minterme,* tandis que les lignes représentent des conditions *Maxterme*, i.e. qu'un service peut être exécuté s'il y a une colonne dans laquelle toutes les conditions sont satisfaites. Comme on peut le voir dans le modèle, des *préconditions* peuvent être associées aux *inputs,* ce qui peut être utilisé comme opérande dans ces conditions rationnelles ou logiques. Par exemple, une *précondition* pourrait être que le produit fournit un modèle 3D à la ressource de manière à ce qu'elle puisse se déplacer autour. De plus, le *ProfMServ* fournit de l'information sur le type de retour (informationnel) qui peut être donné de l'exécution du service. Cette information est fournie en tant qu'*outputs*.

Dans un HMS, un cloud local de *MServices* est créé par les HR avec la publication de tous leurs *ProfMServs*. Ce cloud est disponible pour les autres holons du système de telle sorte que les HP puissent trouver les candidats potentiels à satisfaire leurs besoins. Cette étape est identifiée dans l'architecture de référence holonique HCBA (Chirn and McFarlane, 2000) en tant que « Intégration Statique ». Les candidats potentiels sont évalués par une association de paramètres et attributs entre les *ProfMServs* et les *SpecMServ,* de manière à ce que plus tard ils puissent être interrogés sur leur disponibilité pour pouvoir créer les contrats de service. Dans HCBA, l'acte de découverte et d'établissement des contrats correspond à l'étape de « Intégration Dynamique ».

2.5.5 Modèle *ImpMServ*

La perspective d'*Implémentation de MServices* est unique à chaque fournisseur de service. Elle contient toutes les informations sur les méthodes et technologie utilisées par un HR (fournisseur de service) pour produire les transformations spécifiées par les perspectives Type et *Profil*, i.e. toutes les informations sur comment un service est censé d'être réalisé. Comme illustré sur la Figure 30 avec la perspective *d'implémentation, un MService* peut être composé d'une ou plusieurs *MethDeProces* (Méthode de Processus) et par une collection de *SpecParam*s. Une *MethDeProces* représente une action ou une structure d'actions qui transforment le produit et/ou l'environnement comme décrit par le *Type*. Les *MethDeProces* peuvent être classées en trois types de modèles de processus en fonction des relations internes à leur composition : *Modele_ProcesProd* (modèle de *Processus-Produit*), *Modele_ProcesDisp* (modèle de *Processus-Dispositif)* et simple *AdresseLogique* Les deux premiers implémentent des *MServices* composés, qui sont des processus composés d'autres services à la granularité plus faible. Le dernier représente un programme dans le contrôleur du fournisseur exécutant un *MService* atomique. Comme l'indique sa cardinalité, un HR peut posséder plus d'une méthode réalisant les mêmes transformations. Chacune de ces méthodes a son propre ensemble de spécifications d'attributs, utilisé pour évaluer son éligibilité en comparaison avec d'autres méthodes.

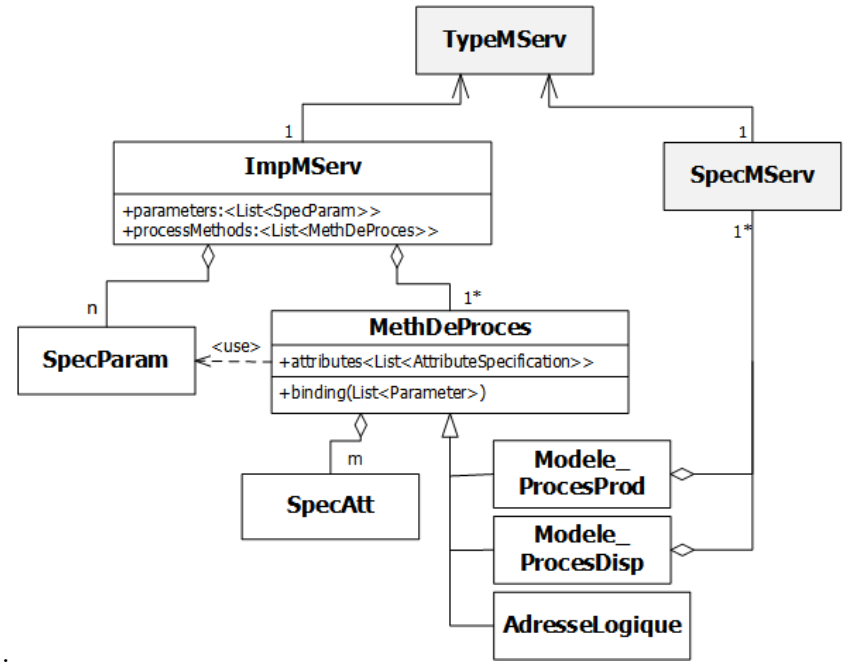


Figure 30 Modèle d'Implémentation de MService

Etant donné que les méthodes de processus peuvent représenter des services composés, les services composés inclus dans les méthodes nécessitent la spécification de leurs propres paramètres. Pour cela, chaque méthode a une fonction d'association utilisée pour générer les spécifications de paramètres des services la composant à partir des spécifications de paramètres du niveau de composition supérieur.

2.6 « *MATCHING* » DES *MSERVICES*

Le "Matching" des services en utilisant une ontologie sémantique telle que OWL-S se fait en employant l'inférence plutôt que des correspondances de type un-à-un pour découvrir les services qui peuvent être réalisés en se basant sur un ensemble des compétences et non sur un type de ressource. Dans cette approche, les *MServices* ne sont pas spécifiques à une ressource, mais sont au contraire des descriptions abstraites des opérations basées sur la nature de leurs effets sans considérations d'aucune ressource ou méthode. Les clients comme les fournisseurs des services définissent leurs spécifications de produits et capacités respectivement, en accord avec une ontologie de services d'application contenant la description de tous les types de MServices. Le «matching » se fait entre une *SpecMServ* et un *ProfMServ* basés sur :

- Le *TypeMServ* de la spécification et du profil ;
- Les valeurs des paramètres de la *SpecMServ*, qui doivent être cohérents avec l'ensemble des profils de paramètres contenus dans le *ProfMServ* exposés par une ressource.

Avec cette approche, il n'y a pas besoin de décrire des effets et préconditions avec un langage interprétable au automatiquement, car la description fonctionnelle d'un *TypeMServ* a pour seul objectif l'interprétation humaine. A la fois les clients et les fournisseurs font référence à la même *OdApp*, donc le « matching » des effets et préconditions de service est implicite, ce qui facilite la découverte des MServices, réduisant ainsi la charge computationnelle pour une implémentation en ligne.

2.7 PROCESSUS ORIENTE-SERVICES (*SOP*)

Dans la section précédente, un modèle conceptuel et informationnel des services de production a été présenté, décrivant les éléments que comporte leur description et ainsi représentant des activités de production à différents niveaux de granularité. Rappelant le modèle conceptuel présenté section 2.3.2, un *MService* possède la propriété de récursivité, qui lui permet de modéliser des services complexes construits par un ensemble de *MServices* plus granulaires. Cette section se concentre sur la composition des processus de production à partir d'un ensemble de MServices, leur encapsulation et présentation au système en tant que *MService,* s'ajoutant à *l'OdApp* en question. Un processus de fabrication, depuis la perspective des fournisseurs, peut être caractérisé par :

- La granularité de ses opérations ;
- L'ontologie avec laquelle il est défini ;
- La simultanéité ou parallélisme entre les opérations qui le compose.

Comme indiqué dans la Figure 31 , un processus peut être classifié selon la fragmentation de sa composition entre service atomique et service composé. Les *processus atomiques* sont des opérations qui peuvent être exécutées avec une seule interaction et peuvent être directement invoqués, comme un simple appel à une fonction. Les opérations qui les forment ne peuvent pas être modifiées logiquement ni invoquées de manière séparée. Les *processus* composés peuvent être identifiés en deux types selon les relations entre les opérations qui déterminent leur structure. A l'intérieur de ce type de processus, la nature des relations entre les opérations organise les processus en deux classes : les *Processus-Produit* et les *Processus-Dispositif*. La distinction est basée sur la possibilité de simultanéité entre les opérations.

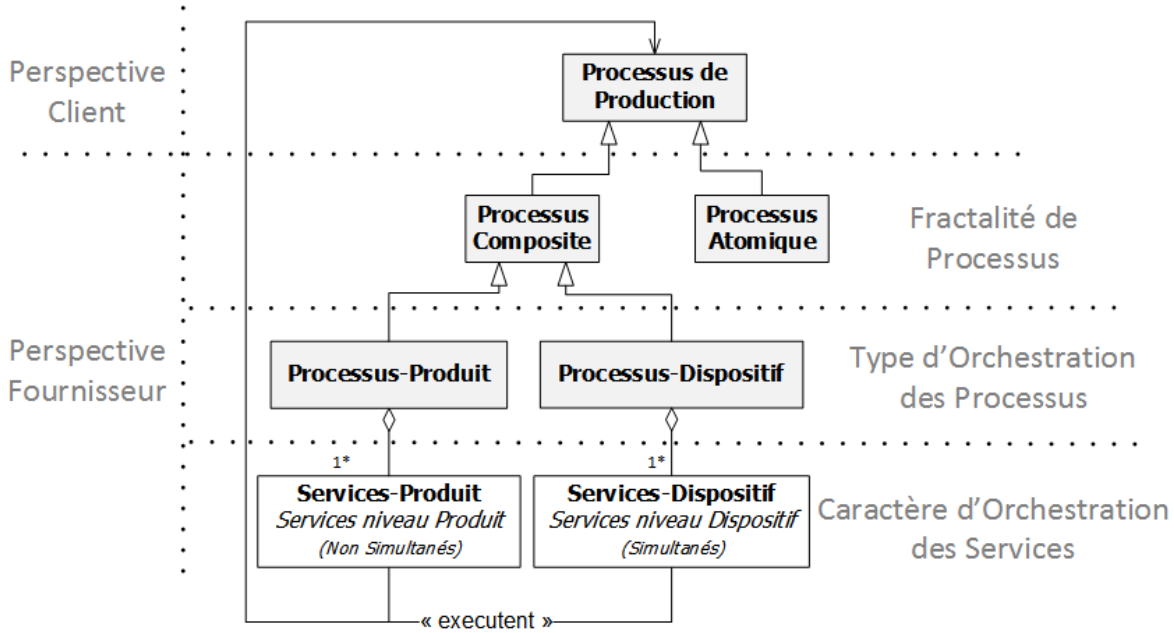


Figure 31 Granularité des processus de production

Les *Processus-Dispositif* sont des opérations de type multiserveur nécessitant des protocoles à pas multiples, i.e. nécessitant plus d'une ressource pour leur exécution. Ces processus peuvent être décomposés en d'autres processus atomiques ou composés, comme l'indique leur cardinalité. Normalement offerts au niveau des postes de travail, là où différentes ressources partagent un même espace de travail, elles comprennent des opérations parallèles (simultanées) ayant un couplage plus étroit. Ceci implique des besoins de synchronisation parmi les ressources et service concernés.

Les *Processus-Produit,* de la même manière que les *processus-dispositif*, sont composés par des opérations multiserveur à pas multiples mais de nature non simultanée, c'est-à-dire que pas plus d'un service n'est exécuté en même temps (donc séquentiellement). Ce type de processus caractérise le flux de travail dans les systèmes avec configurations de type Job-Shop et dans des systèmes contrôlés par le produit, où le produit se déplace d'un poste (machine) à l'autre pour subir des transformations. Il est nommé ainsi car il est construit autour du produit et de son parcours au travers du système de production.

Comme le montre le bas de la Figure 31 , la récursivité des processus est conservée à la fois pour les MServices-Produit et les MServices-Ressource. Comme le diagramme l'indique, ces services exécutent des opérations de production, lesquelles peuvent être formées par des opérations atomiques ou par des opérations composées, répétant ainsi la structure récursive descendant jusqu'à des opérations atomiques et en remontant jusqu'aux *MServices* offerts aux clients en tant qu'ordres de production.

2.7.1 Modèle de *Processus-Produit* orienté services (*ProcesProd-S*)

La principale caractéristique des processus au niveau du produit est l'occurrence non-simultanée des services. Ceux-ci sont décrits dans le modèle conceptuel présenté dans la Figure 32 Le modèle conçu est basé sur la norme ISA SP-95 (ISA-95, 2000), qui spécifie toutes les informations requises pour la production d'un produit. Ce modèle regroupe les informations d'une manière adaptée à répondre aux principes de SoA, les systèmes contrôlés par le produit et la spécification des produits personnalisables. Effectivement, un processus-produit a pour objectif d'être piloté par l'un des acteurs représentant un produit.

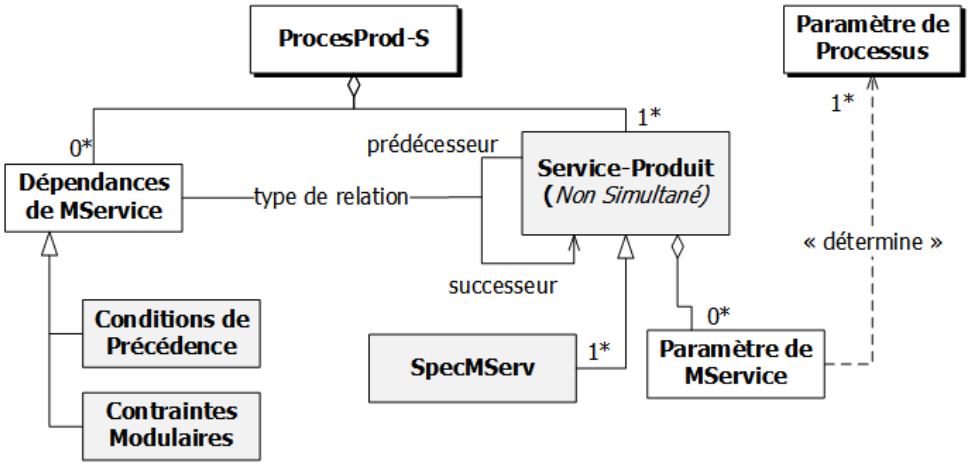


Figure 32 Modèle conceptuel de Processus-Produit orienté Services

Un processus-produit est donc composé par un ensemble de MServices-Produit représentant des opérations non-simultanées. Chacun de ces services comporte un ensemble de paramètres-service pour délimiter la portée des transformations, selon le modèle conceptuel présenté dans la section 2.3.2. Ceux-ci, appartenant à un *MService* d'ordre inférieur au processus-produit qu'il compose, sont issus grâce à une fonction de transformation à partir des paramètres-processus d'un ordre plus haut décrivant la portée des transformations fixées par le client du processus.

La partie la plus pertinente du modèle, celle qui permettra de déterminer la méthode d'orchestration des séquences de production, est la façon dont les interdépendances parmi les services sont exprimées. Dans ce cas (celui du processus-produit), les interdépendances sont déclarées dans une perspective de prédécesseur, avec un tableau de conditions de précédence. Des contraintes modulaires sont ajoutées

dans le tableau de précédences pour intégrer une personnalisation modulaire dans la composition du processus. Plus des détails sur la définition des dépendances, sera présenté dans le Chapitre 5.

2.7.2 Modèle de *Processus-Dispositif* orienté services (*ProcesDisp-S*)

Dans un processus au niveau équipement, les services peuvent être concurrents, et donc avoir des relations plus étroitement couplées entre eux. Par exemple, une cellule de production comportant différents bras robotisés peuvent opérer simultanément sur un produit, l'un maintenant le produit en position pendant que l'autre effectue une opération de soudage. Les dépendances entre ces types d'opérations ne nécessitent pas seulement la définition de relations de précédence, mais également de synchronisation. La structure de processus au niveau équipement peut être définie par des structures de contrôle telles que celles que l'on peut trouver dans des langages de programmation ayant une approche orientée évènements pour coordonner l'exécution de services composés.

Dû à la nature très couplée de ses opérations, un *processus-dispositif* doit être piloté par un *Holon Ressource* muni de méthodes portant la logique de coordination et de synchronisation. Comme indiqué précédemment, un *ProcesDisp-S* a une structure très similaire à celui des *ProcesProd-S*.

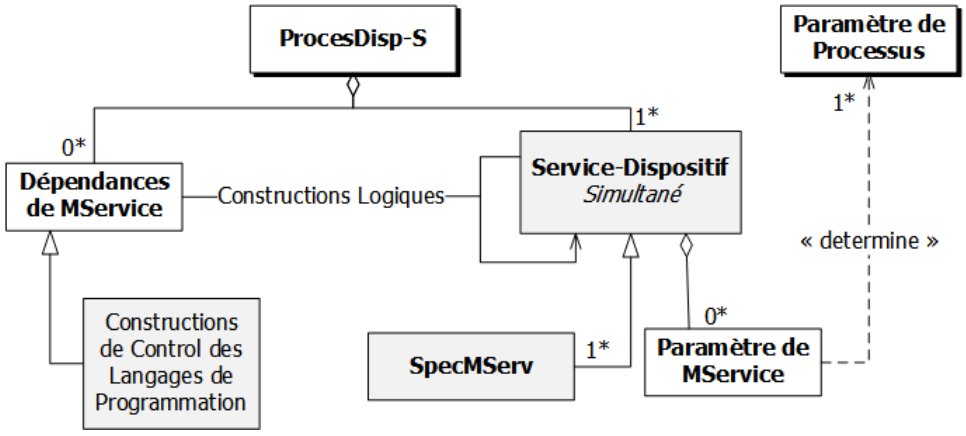


Figure 33 Modèle de processus Processus-Dispositif orienté services

Un *ProcesDisp-S* est composé par un ensemble de *MServDisp* (*MServices niveau Dispositif*) qui sont caractérisés par des opérations qui peuvent s'exécuter de manière simultanée. Du fait des possibles interactions entre ces services, une liste des dépendances est aussi spécifiée. Contrairement aux *SoP*s, une liste de contraintes de précédence n'est pas appropriée pour synchroniser des activités concurrentes. Ceux-ci doivent se définir avec un langage ayant des constructions adaptées à la description des relations conditionnelles et du parallélisme, comme celles trouvées dans la plupart des langages de programmation conventionnelles. Cependant, les contributions de cette thèse ne concernent pas à la définition des *ProcesDisp-S* mais la conception, planification et ordonnancement des *ProcesProd-S*.

2.8 CONCLUSION

Ce second chapitre avait pour objectif de présenter l'adaptation du concept de services issu des SoA aux caractéristiques des systèmes de production. D'une étude de l'existant dans la littérature est ressorti le besoin d'une adaptation des services pour la représentation des activités de production, accueillant de plus la récursivité voire la fractalité typique des structures des produits et systèmes de production.

Après avoir présenté le contexte et les caractéristiques fondamentales des activités de production, nous avons proposé le concept de *MService,* permettant une unification des domaines de la production et des SoA. Ce concept a été formellement modélisé et positionné dans son écosystème. De là, le besoin de la

définition plus stricte de la sémantique du système complet nous a amené à proposer l'utilisation d'ontologies d'application (ou de domaine lorsque disponible) pour la définition des services. Un ensemble de perspectives a également été défini, permettant de mettre en avant le rôle central du service dans le système de contrôle, ce qui l'amène à avoir plusieurs représentations différentes selon l'acteur avec lequel il interagit. Le prochain chapitre nous permettra de mettre en perspective l'utilisation de ces services au sein d'une architecture holonique et les modifications que cela entraine dans la conception des relations entre ces holons.

Enfin, nous introduisons un cadre de modélisation afin d'intégrer le concept de services au sein des gammes de production avec pour objectif l'ordonnancement et la planification des ordres de production. La modélisation de gammes sera développée intégralement au sein du Chapitre 4 et l'orchestration de ces gammes sera étudiée au Chapitre 5.

Chapitre 3

Système Holonique de Production Orienté-Services (*SoHMS*) : Un Nouveau Paradigme

Le chapitre précédent a permis de proposer un modèle générique de services de production. L'idée générale de ces travaux est de coupler ce concept de services et les architectures de contrôle classiquement rencontrées pour le contrôle de systèmes de production afin d'augmenter la flexibilité structurelle dont l'architecture pourra tirer parti et s'adapter aux aléas qu'elle rencontre. Parmi ces architectures, les architectures holoniques ont été présentées comme répondant aux exigences et sont de bons candidats pour ce couplage.

Ce chapitre a pour objectif de présenter le couplage de l'orientation-services avec les systèmes de production holonique, ce qui donne le jour au concept de système de production holonique orienté-services (SoHMS). Le chapitre démarre par une présentation des atouts de ce couplage et une délimitation du type de systèmes potentiellement concernés. Une méthodologie de conception est ensuite présentée, suivie d'une proposition d'intégration de la notion de services dans la définition des caractéristiques et fonctionnalités de chacun des holons de base de notre architecture. Enfin, l'impact des services sur les relations entre holons à l'intérieur de l'architecture de contrôle est présenté.

3.1 INCORPORATION DES *MSERVICES* AUX SYSTEMES HOLONIQUES

L'incorporation des concepts de service dans les systèmes holoniques donne naissance à un nouveau type de paradigme, celui des Systèmes Holonique de Production Orientés-Services (SoHMS - *Service Oriented Holonic Manufacturing Systems*). Une telle intégration fait alors du *MService* le principal élément de description et de négociation dans le HMS, ce qui a pour effet de résoudre leurs problèmes d'interopérabilité et d'intégration. Cette fusion se révèle être très attractive car elle combine les avantages en termes de flexibilité des deux architectures : la flexibilité au niveau du contrôle et au niveau des processus.

Cette intégration devient possible grâce à la similarité trouvée à certains points des deux architectures :

- **Répétabilité et Réutilisation** *:* Les deux paradigmes présentent des principes de modélisation des structures répétitives pour ainsi motiver leur réutilisation à travers l'architecture (les services pour SoA ; les ressources pour HMS).

- **Fractalité et Récursivité** : Les services autant que les holons sont des structures fractales et récursives. Fractales, car elles peuvent être décomposées en services/holons plus granulaires, et récursives parce qu'on peut trouver les mêmes structures répétées tout au long de sa composition. Un service comme un holon peut être composé par d'autres services ou d'autres holons, respectivement, ayant le même modèle.
- **Encapsulation** *:* SoA utilise les services comme une interface représentant un processus de transformation tout en encapsulant toutes les subtilités que l'on peut rencontrer lors de son implémentation. De la même façon, les systèmes holoniques utilisent les holons pour encapsuler la complexité d'une holarchie, l'holarchie pouvant être représentée par un holon.

La combinaison de ces deux paradigmes apparait comme une option très attrayante comme ont pu le démontrer des travaux tels que (Bellifemine et al., 2007; Jammes et al., 2005; Jammes and Smit, 2005) ou plus récemment (Morariu et al., 2013). La flexibilité obtenue a des origines qui sont bien décomposées, entre le HMS au niveau structurel pour l'assignation des taches et les associations entre entités (holons) au sein de l'architecture de contrôle, et la SoA au niveau processus pour la décomposition et l'encapsulation de processus qui permet leur distribution entre les ressources. Dans un *SoHMS*, toutes les opérations de production peuvent être représentées par un *MService* qui peut être exécuté sur un produit et peut être offert par une ou plusieurs ressources du système. Grâce au modèle de MServices, toute opération dans une application peut avoir une identification et une description propre qui correspond aux caractéristiques de leurs transformations. Cette représentation s'adapte bien à la répétitivité que l'on peut trouver dans un système de production, soit dans la spécification des processus de production soit dans la spécification des capacités d'une ressource.

Du fait que les descriptions des opérations sont déterminées en termes de transformations agrégées sans égard aux méthodes utilisées pour leur implémentation, elles peuvent être standardisées pour accueillir différentes méthodes ayant les mêmes effets de transformation. Le *MService* étant une représentation des effets d'une opération peut donc être réutilisé pour décrire et représenter d'autres méthodes, d'où une réutilisation du type de *MService* en tant que carte de présentation. C'est ainsi que le service sert d'interface entre les niveaux d'implémentation et les niveaux de contrôle.

La réutilisation apparait aussi lors de la spécification des processus. Les processus de production sont construits en fonction d'une collection de *MServices* selon une ontologie d'application. Un processus de production peut réutiliser des méthodes ayant été implémentées originellement pour d'autres processus de production mais ayant la même description des effets du *MService,* d'où une réduction des efforts de reprogrammation. Ce dernier point devient important lors de l'intégration de nouvelles technologies au sien d'une ligne existante. La représentation des opérations par des *MServices* permet une séparation complète de la spécification des processus de production de la partie des connaissances relatives à l'outil de production. Une spécification produit peut donc être implémenté dans n'importe quelle plateforme *SoHMS* tant qu'elle implémente la même ontologie d'application (mêmes termes et concepts descriptifs).

Les HMS gagnent aussi en interopérabilité et en intégration grâce à l'adaptation des services. L'intégration est facilitée par l'adoption des *MServices* comme élément descriptif. Les ressources étant des porteuses de capacités de transformations peuvent être représentées par la collection des *MServices* que chaque ressource offre au système. Les capacités des différentes ressources implémentant les différentes technologies peuvent être intégrées au système de pilotage très facilement grâce à l'encapsulation de leurs méthodes. L'intégration des systèmes hérités est considérablement facilitée en créant un holon virtuel gérant l'utilisation de la ressource et offrant ses capacités sous forme de MServices, isolant les informations qui ne sont pas relatives aux activités de pilotage de leur contrôle local. En outre, l'intégration de nouvelles capacités est aussi facilitée avec la modification, ajout ou suppression de *MServices* à des ressources déjà existantes.

Quant à l'interopérabilité, le modèle de *MService* avec la définition d'une ontologie d'application implique une structure de communication sur laquelle les holons peuvent baser les négociations et répartir la charge de responsabilité de production. Le modèle de *MService* propose la syntaxe et la structure des données, tandis que l'ontologie d'application (*OdApp*) représente une sémantique unifiée des éléments de description d'un type spécifique de *MService.* Ainsi, pour atteindre l'interopérabilité dans un *SoHMS*, il suffit de concevoir une ontologie de services sur laquelle on pourra construire des spécifications de produit et la spécification des ressources : besoins et capacités sont alors décrits en se basant sur les mêmes éléments. Une spécification produit construite en se basant sur une *OdApp* spécifique peut être lancée dans n'importe quelle autre plateforme qui implémente la même ontologie. Ceci n'est pas limitant, car les méthodes d'une ressource peuvent avoir plusieurs représentations appartenant à différentes ontologies.

Du fait que le *MService* est l'élément de description des activités de l'atelier, il devient ainsi aussi l'élément de base des négociations holoniques. Tous les modèles des holons et des autres acteurs, les mécanismes de coordination, les protocoles de négociation et les interactions (planification de processus, ordonnancement des opérations, allocation de ressources, détection de défaillances, entretien des ressources, lancement des ordres, etc.) sont conçus sur la base du modèle de *MServices* : c'est que l'on appelle les *négociations orientées MServices.* Ces négociations se caractérisent par les points suivants :

- Il existe un holon (client) nécessitant un service qui peut être réalisé par un holon (fournisseur). La négociation entre ces deux éléments se conclut avec l'établissement d'un contrat, établi sous des termes et conditions spécifiques.
- Les critères de négociation peuvent porter sur plusieurs sujets relatifs aux caractéristiques du service qui sont importants pour le client, tels que la qualité de service (QoS), le coût, le temps de traitement, la consommation d'énergie, la fiabilité du fournisseur pour offrir le service avec succès, etc.
- Les négociations peuvent être itératives sous plusieurs tentatives jusqu'à l'attribution d'un contrat. Si le nombre de tentatives atteint un seuil ou bien une échéance temporelle de décision est dépassée, alors l'exécution d'une action par défaut change les conditions de négociation.
- Un service peut être offert par plus d'un fournisseur dans le système. Les services peuvent être identiques dans leurs caractéristiques ou peuvent varier en échelle des dimensions de ses transformations.
- Un holon peut être client et fournisseur de services.
- Le temps de négociation est un facteur important ou critique. Il est important que le modèle de négociation soit conçu autour des préceptes suivants :
    - Le temps pour atteindre une conclusion (une solution) doit être limité ;
    - Le temps de négociation peut être important pour certaines négociations et critique pour d'autres. Le modèle de négociation doit considérer ce paramètre pour assurer la conclusion des négociations pour une échéance cohérente ;
    - Le temps de négociation doit, dans tous les cas, être inférieur à la durée séparant le début du processus de négociation de la date à laquelle la décision qui en découle sera appliquée.

L'incorporation du concept de services n'implique pas seulement l'adaptation du modèle de services aux besoins et caractéristiques des processus de production. Elle implique aussi une réadaptation des acteurs de l'architecture de contrôle ainsi que des activités de contrôle autour du modèle de MServices. Les prochaines sections de ce chapitre nous permettront de redéfinir une architecture holonique dont la

description des holons se fait en rapport aux services en termes de besoin et capacités en vue de préserver une architecture fractale et récursive.

3.2 *SoHMS* : UNE METHODOLOGIE DE MODELISATION

Comme (McFarlane and Bussmann, 2003) l'a remarqué lors de ses travaux pour la conception des systèmes de contrôle basés sur des agents : pour qu'une technologie soit bien acceptée dans l'industrie, celle-ci doit fournir les méthodologies nécessaires pour la conception de systèmes basés sur cette technologie. Ceci s'ajoute à la liste des raisons expliquant l'adoption tardive des HMS et des technologies multi-agents dans l'industrie, malgré les bons résultats de ces systèmes dans le domaine de la recherche académique. Toujours selon (McFarlane and Bussmann, 2003), ces méthodologies doivent satisfaire deux points importants :

- **Des modèles appropriés :** Les modèles proposés par une méthodologie doivent être dûment et clairement liés aux concepts du domaine d'application (ici la production). Les modèles initiaux doivent être conçus sur la base de ces concepts de domaine et sont extensibles à condition que les nouveaux concepts et/ou modèles soient basés sur ceux déjà introduits.
- **Une méthode prescriptive** : Les méthodes doivent être descriptives, c'est-à-dire qu'elles doivent détailler chaque étape que le développeur doit suivre, et pour chaque étape elles doivent aussi indiquer quelles sont les tâches à réaliser par le développeur et illustrer comment chaque étape doit être réalisée.

Le *SoHMS* proposé dans cette thèse se présente alors (de manière analogue à la méthodologie DACS par (McFarlane and Bussmann, 2003)) comme une méthodologie de référence pour la modélisation des systèmes de production, combinant les principes des paradigmes holonique et orienté-services tout en reprenant certains concepts les plus connus et acceptés des architectures de référence holoniques existantes, notamment PROSA et ADACOR, pour la conception de systèmes de production flexibles. Les apports de cette thèse représentent une base offrant la définition des modèles des holons et un catalogue des protocoles représentant le *backbone* du système, comme des méthodologies pour la spécification du système en termes de spécification des produits, des ressources et d'une ontologie-services.

Ainsi donc, les modèles de l'architecture *SoHMS* et leurs rôles sont conçus sur la base du modèle de *MServices* ce qui est en accord avec la première exigence de modélisation de (McFarlane and Bussmann, 2003). Quant aux méthodologies prescriptives, cette thèse présente des méthodologies pour la création des *OdApp*s, la spécification des produits et leurs gammes de production et la spécification des ressources sur la base des mêmes concepts et de manière prescriptive. De plus, une stratégie de planification et d'ordonnancement, conçue sur la base des modèles du *SoHMS*, est proposée sous la forme de comportements pouvant être ajoutés à l'architecture comme une extension pouvant être remplacée sans altérer les autres fonctionnalités du système.

L'implémentation d'un *SoHMS* dépend alors uniquement de l'identification des types de ressources, la granularité de ces dernières et de leurs services, et la définition de l'ontologie *OdApp*. Le *SoHMS* peut être implémenté en déclarant ces informations sous forme de fichiers XML avec la programmation d'un*e* interface logicielle que l'on dénomme *SIL* pour *« Service Interface Layer »*, fonctionnant comme une interface entre les instances des *MServices* et les commandes envoyées à l'atelier pour l'exécution des méthodes correspondantes (la généralisation et la méthodologie de conception des modèles *SIL* pour leur déclaration avec des fichiers XML fera partie des perspectives de ce travail).

3.3 TYPE DE SYSTEMES D'APPLICATION

Du fait du grand spectre d'application des *SoHMS*, il n'existe pas de système universel qui s'adapte à tout et se positionne comme une solution la plus adéquate en général. Pour ce motif, avant de continuer avec la présentation du *SoHMS* , cette section est dédiée à la description du contexte d'application choisi, pour bien placer le *SoHMS* développé dans son contexte et pour bien expliciter le fait que les concepts sont applicables à une gamme de systèmes plus ouverte que le système spécifique présenté en illustration au chapitre 6. Ces systèmes peuvent avoir des caractéristiques variées quant à leur composition physique, les types de produits, les types de processus et le modèle de production (Figure 34).

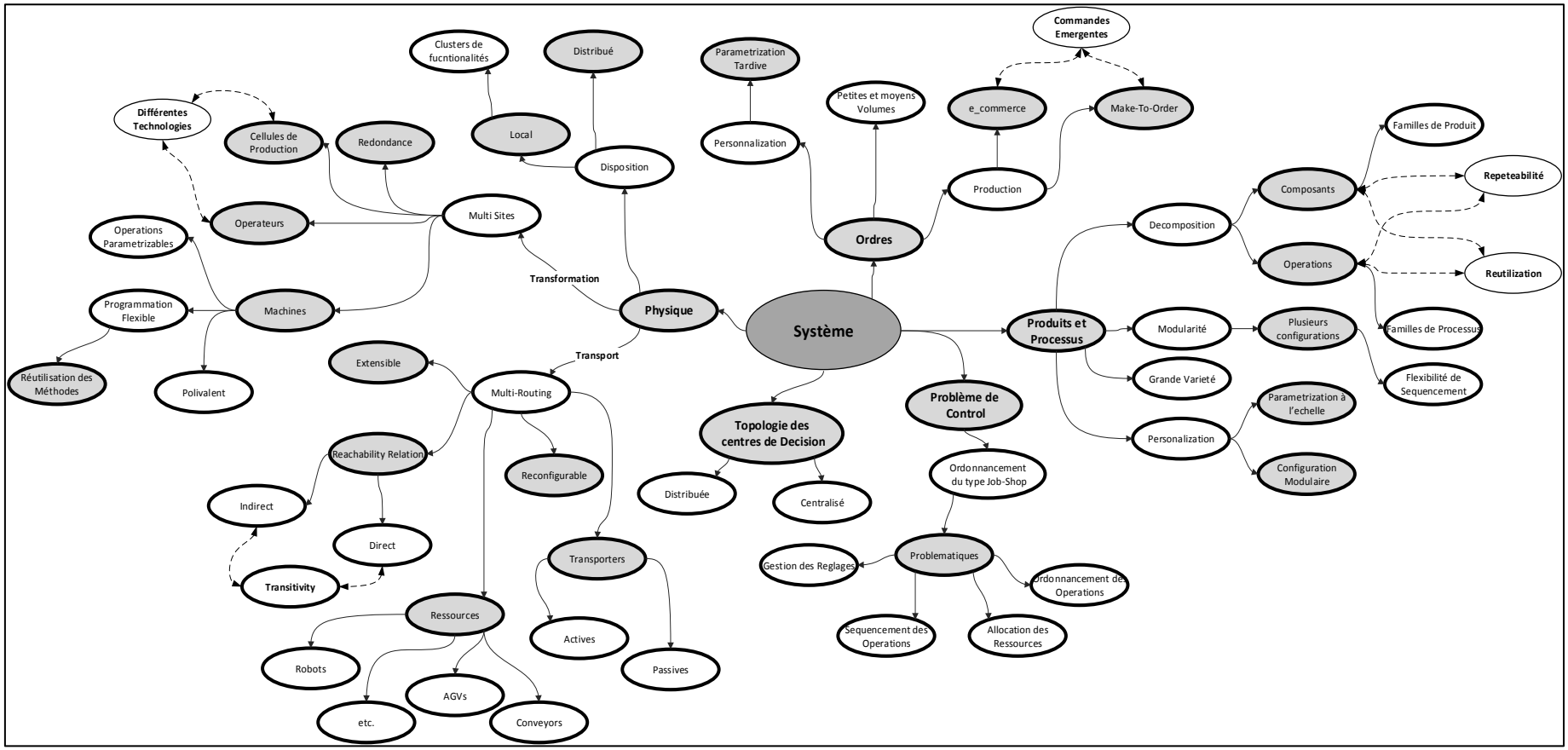


Figure 34 Caractéristiques du système d'application

Le problème de contrôle

Les protocoles de contrôle (planification et ordonnancement) ont pour objectif de donner une solution aux problèmes de type Job-Shop Flexibles (FJS). Le FJS est une extension du problème de Job-Shop classique où l'on suppose l'existence d'une seule machine capable d'exécuter une opération spécifique. Le FJS, en revanche, considère une configuration plus flexible où une opération peut être exécutée par plus d'une ressource. Les contraintes sont : les *jobs* sont formés par une séquence de différentes opérations de production exécutables par une ou plusieurs ressources disponibles dans l'atelier de production. Leur séquence d'exécution peut varier, ne pas solliciter toutes les ressources et avoir des routages multiples. Ce type de systèmes a son origine dans l'apparition des équipements permettant des changements rapides d'outils avec des contrôleurs programmables leur procurant un certain degré de flexibilité. Par conséquent, parmi les problématiques présentées au système de contrôle, on trouve la gestion des réglages, l'allocation des opérations aux ressources et le séquencement des opérations pour chaque *job*.

Le système physique

Le système physique est composé par un groupe de ressources de transformation et un groupe de ressources de transport. L'outil de production est caractérisé par un système multi-sites (multipostes) ayant de la redondance dans l'offre des opérations. Les sites peuvent représenter tout type de technologie de production comme des cellules de production, des machines ou des opérateurs humains dans des stations manuelles ou semi-automatisées. Les machines comme les cellules de production peuvent présenter des capacités de production polyvalentes et de la programmation flexible. Chaque site de production compte avec les matières premières ou sous-produits nécessaires à l'exécution d'une opération permettant l'implémentation d'une stratégie du type « Assembly-to-Order » (ATO) et/ou « Make-To-Orde »r (MTO).

Le groupe des ressources de transport forme un système de transport donnant de l'interconnectivité entre les différents sites de l'outil de production. Dû à la caractéristique des FJSs d'avoir plus d'une séquence de production, le transport est considéré Multi-Routage, où les produits peuvent suivre des routages différents parmi les sites de production. Deux sites peuvent avoir une relation d'atteignabilité soit directe (une ressource seule a la capacité de déplacer un produit de son origine vers sa destination) ou bien indirecte par transitivité (nécessite plus d'une ressource pour transporter un produit à sa destination finale). Le système peut ne pas présenter de relation d'atteignabilité complète entre tous les sites.

Les ressources de transport peuvent être de tout type, tant qu'elles sont capables de déplacer le produit d'un site à un autre, par exemple : des convoyeurs, des humains, des AGVs, etc. De plus, le système de transport est aussi composé par un groupe de transporteurs qui prennent le rôle de véhicules porteurs du produit pour s'insérer dans le système de transport (par exemple des palettes). Ces transporteurs peuvent être passifs ou bien actifs, les premiers comptant avec un système d'auto-identification comme des étiquettes RFID, le second ayant en plus les capacités d'appliquer physiquement les décisions qu'il prend (actionneurs) (Gamboa Quintanilla et al., 2013a).

Finalement, du aux besoins d'adaptabilité pour répondre aux besoins de situations changeantes, on considère un système ayant possiblement des changements fréquents dans sa configuration. De tels changements comprennent l'ajout, le retrait ou la substitution de nouveaux sites et de nouvelles ressources dans un site, ou la reconfiguration des routes du système de transport en ajoutant, supprimant, ou modifiant des connexions entre les sites.

Les produits et processus

Les applications spécialement visées sont celles fabriquant des produits ayant une grande modularité, capables d'être décomposés en plusieurs composants avec une grande granularité. L'offre des produits de l'application peut être catégorisée en plusieurs familles de produits, dont l'ensemble des produits représente une grande variété de produits individuels, donc un grand nombre de combinaisons de caractéristiques possibles. Quant à leurs processus de production, les mêmes principes s'appliquent. Au niveau de l'atelier, on peut trouver un grand nombre d'opérations (services) et un grand nombre de procédures peuvent être regroupées en familles de processus (correspondant à la réalisation d'une famille de produit). Les deux familles peuvent être personnalisables au travers du paramétrage des caractéristiques de leurs composants et opérations au niveau d'échelle comme au niveau configurationnel.

Les processus de production, dû à leur modularité et aux relations entre leurs modules, peuvent montrer des niveaux hauts ou bas de flexibilité dans leur séquencement, menant à plusieurs séquences possibles qui aboutissent au même résultat final. De plus, on considère les caractéristiques suivantes pour les processus de production :

- Tous les opérations sont non-préemptives (une fois l'opération initiée, elle ne peut pas être interrompue sauf si le produit est à considérer défectueux) ;
- Il n'y a pas de parallélisme dans l'exécution des services pour un produit (en accord avec le processus-produit) ; du point de vue du produit, un seul service est exécuté à la fois. Le parallélisme est seulement présent de manière indirecte avec la production des sous-produits dont la production est gérée par les ressources offrant leur assemblage.

Du fait de la grande variété de variantes de produits et à la production de produits individuels, tout produit en cours de production doit compter avec un système d'auto-identification, tel que les codes-barres, les étiquettes RFID, identification unique pouvant être communiquée au système pour leur routage et traitement.

L'environnement de production

L'environnement de production est caractérisé par les stratégies de production de *push-pull* (production poussée-tirée) ou bien *Make-To-Order* (MTO, production à la demande), où le point de découplage de la production poussée retombe au niveau des étapes d'assemblage. Le degré de stratégie MTO peut être celle d'une stratégie *Assemble-to-Order* (ATO ; assemblage à la demande) ou d'une stratégie *Build-To-Order* (construction à la demande), où les composants sont déjà disponibles pour leur assemblage ou bien où leur production est à lancer à l'arrivée de nouveaux ordres. Ces stratégies sont normalement dédies aux applications avec des produits coûteux à stocker ou à volumes faibles.

Les ordres peuvent arriver à tout moment lors de la production, demandant des estimations sur les dates de livraison. Ces ordres émergents donnent au système un comportement dynamique, changeant son état à chaque nouvelle arrivée d'ordre. Les ordres de production peuvent être considérés en petits lots ou comme des produits individuels. Différentes perturbations peuvent se présenter au cours de la production, telles que les pannes machines, l'indisponibilité d'un outil ou composant, ou bien des délais d'opérations de production entrant en conflit avec les plans des autres produits. Tout ce dynamisme demande au système de contrôle de recalculer ses plans à chaque changement de son état.

3.4 ETAPES DE LA METHODOLOGIE

Identification des agents holons

La première étape de la méthodologie consiste en l'identification des holons présents dans le système et la mise en relation avec leurs composants physiques, autrement dit l'attribution des agents virtuels aux éléments physiques Les concepteurs doivent choisir quel est le modèle de Holon le plus approprié pour la représentation de chaque ressource du système. Leur décision doit aussi porter sur la granularité avec laquelle les ressources seront le mieux représentées par rapport à l'application en question.

Définition d'une *OdApp*

La deuxième étape consiste en la description des activités (de transformation, approvisionnement, assemblage, etc.) pertinentes pour l'application. Ainsi, comme pour la première étape, les concepteurs définissant l'*OdApp* devront faire des choix sur la granularité des MServices. Celle-ci peut être relative aux méthodes déjà existantes avec une approche type *bottom-up* ou bien avec une approche *top-down* à partir directement de la description des tâches de l'application.

Sélection des comportements

Dans cette troisième étape, il ne reste qu'à sélectionner parmi les comportements offerts par le *SoHMS* ceux avec lesquels les Holons Produit se lanceront au sein du système de production. Ce choix sera fait en fonction de plusieurs critères, allant de comportements complètement réactifs à des comportements avec des stratégies plus globales ; ces dernières nécessitent généralement des capacités de calcul plus évoluées.

3.5 ARCHITECTURE *SOHMS*

L'architecture conçue se base principalement sur les principes et concepts introduits par l'architecture PROSA, en combinant quelques concepts d'interaction des architectures HCBA et ADACOR. Cette proposition d'architecture reprend les concepts de *Holon Produit (HP)*, *Holon Ressource (HR)*, et *Holon Ordre (HO)* provenant de PROSA, ainsi que le concept de *Directory Facilitator* (*DF*) des architectures multi-agents). Même si les concepts de base restent proches de leurs origines, nous avons adapté leur comportement pour faire du *MService* le principal élément d'échange dans les interactions, orienté vers les activités de planification et d'ordonnancement. La Figure 35 montre le diagramme de classe de l'architecture, les relations et échanges d'informations entre les différents acteurs.

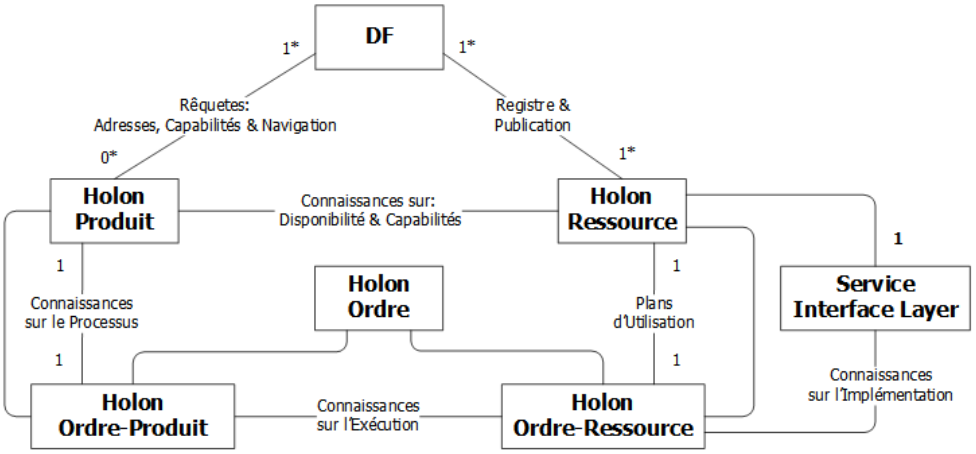


Figure 35 Diagramme de Classes UML de l'architecture *SoHMS*

En haut du diagramme, on peut voir le DF avec les HP et HR. La relation DF-HR consiste en l'enregistrement des capabilités des HR pour les rendre visibles au reste du système à travers un système de « pages jaunes ». Les HP peuvent accéder à cette information au travers du DF, en sollicitant des services d'identification de capacités, d'adressage (directions virtuelles), et d'information sur la navigation (information de transport pour se diriger à un port spécifique). Une fois mis en relation, le HP et le HR échangent directement de l'information sur la disponibilité de la ressource pour inscrire des intentions. A la différence de PROSA, le *oOHMS* compte deux types de *Holons Ordre* : *Holon Ordre-Produit* (*HOP*) et *Holon Ordre-Ressource*(*HOR*). Ces deux types permettent de répondre aux besoins d'exécution des plans (plus d'informations sur cette distinction seront donnés dans la section suivante).

Ainsi, un HP possède un *HOP* à qui il délègue la responsabilité d'exécuter les plans de production issus des négociations du HP. De même, un HR possède un *HOR*, à qui il délègue la responsabilité de répondre aux requêtes d'exécution de services par les *HOP* en contrôlant la satisfaction des plans d'utilisation de HR. Un nouvel élément ajouté à l'architecture est le « *Service Interface Layer* » (*SIL*), servant comme d'interface entre les descriptions de services et les commandes au niveau de l'atelier (réseaux de terrain). Du fait de sa singularité, chaque ressource possède un élément *SIL*, contenant toute l'information sur la manière d'implémenter un service au niveau terrain. Ainsi, le *HOR* envoie tous ses ordres et reçoit toutes les notifications d'exécution au travers du *SIL*. Une définition plus détaillée des rôles et de la répartition des informations de chaque acteur est présenté dans la section suivante.

3.5.1 Les holons

Holon Produit

Le HP (Figure 36) est identique à celui défini dans PROSA. C'est une entité virtuelle, contenant toutes les informations sur la gamme de production du produit en question. Dans *SoHMS*, à la différence de PROSA, le HP n'est plus une entité passive, serveur d'informations, mais une entité active dans les activités de planification et d'ordonnancement du fait qu'il contienne tous les informations de la gamme et devient l'orchestrateur de sa propre production. Il implémente, au travers de la négociation et des mécanismes de coordination, des stratégies pour l'exploration de l'espace des solutions délimité par la gamme du produit, les capabilités disponibles dans l'atelier et leur disponibilité. Le HP est aussi responsable de l'évaluation de ces solutions sur la base d'un critère d'évaluation tel qu'une fonction multi-objectif pondérée. (Rodrigues et al. 2015) apportent des concepts intéressants et novateurs sur ce sujet.

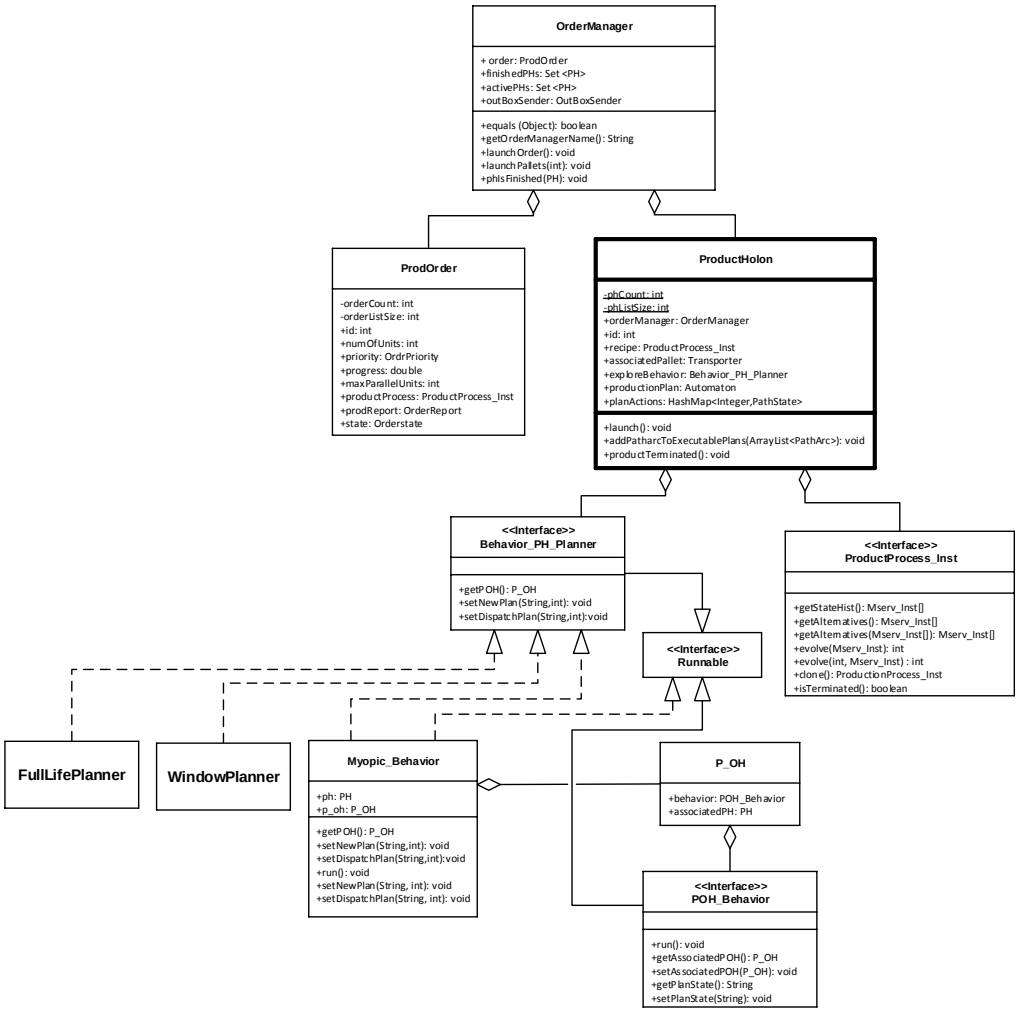


Figure 36 Diagramme de Classes UML : l'Holon Produit

Holon Ressource

Le HR est la partie informationnelle de chaque ressource dans le système de production. Il sert de gestionnaire d'une ou plusieurs ressources (agglomérées) offrant les capabilités de transformations sous forme de *MServices* décrits par leur *Profils*. Son fonctionnement reste assez simple et ressemble à celui d'un agent de réservations répondant aux appels de services des HP demandant des informations sur la disponibilité de la ressource en question. Le HR a la capacité de rechercher à satisfaire ses objectifs en contrôlant ses réponses aux appels. Un HR peut avoir une disponibilité qu'il décide ne pas proposer en réponse à un appel si cela ne satisfait (ou convient) pas à ses objectifs, comme par exemple le besoin d'une période de maintenance conditionnelle. En revanche, tout HR est obligé de proposer une solution pour éviter des situations de blocages au niveau des produits. Ainsi, tout produit doit toujours pouvoir construire une solution faisable, même si elle n'est pas performante. De plus, le HR gère les plannings d'utilisation de ses ports d'entrée (ports qui peuvent être partagés entre plusieurs ressources). Ce planning sert durant la planification des services de transport (détaillée dans le chapitre 6). Les méthodes d'implémentation sont encapsulées par les descriptions de *MServices* et sont accessibles à travers le *SIL*, permettant l'envoi de commandes d'exécution de service et la remontée de notifications depuis le réseau de terrain. La Figure 37 montre les informations portées par le *Holon Ressource* et les opérations qu'il peut exécuter.

Une autre fonction importante est la notification de son indisponibilité, due par exemple à une panne. Pour renforcer la détection des problèmes dans le système et atteindre une meilleure réactivité, le HR doit rafraichir son registre chez le DF de manière périodique.

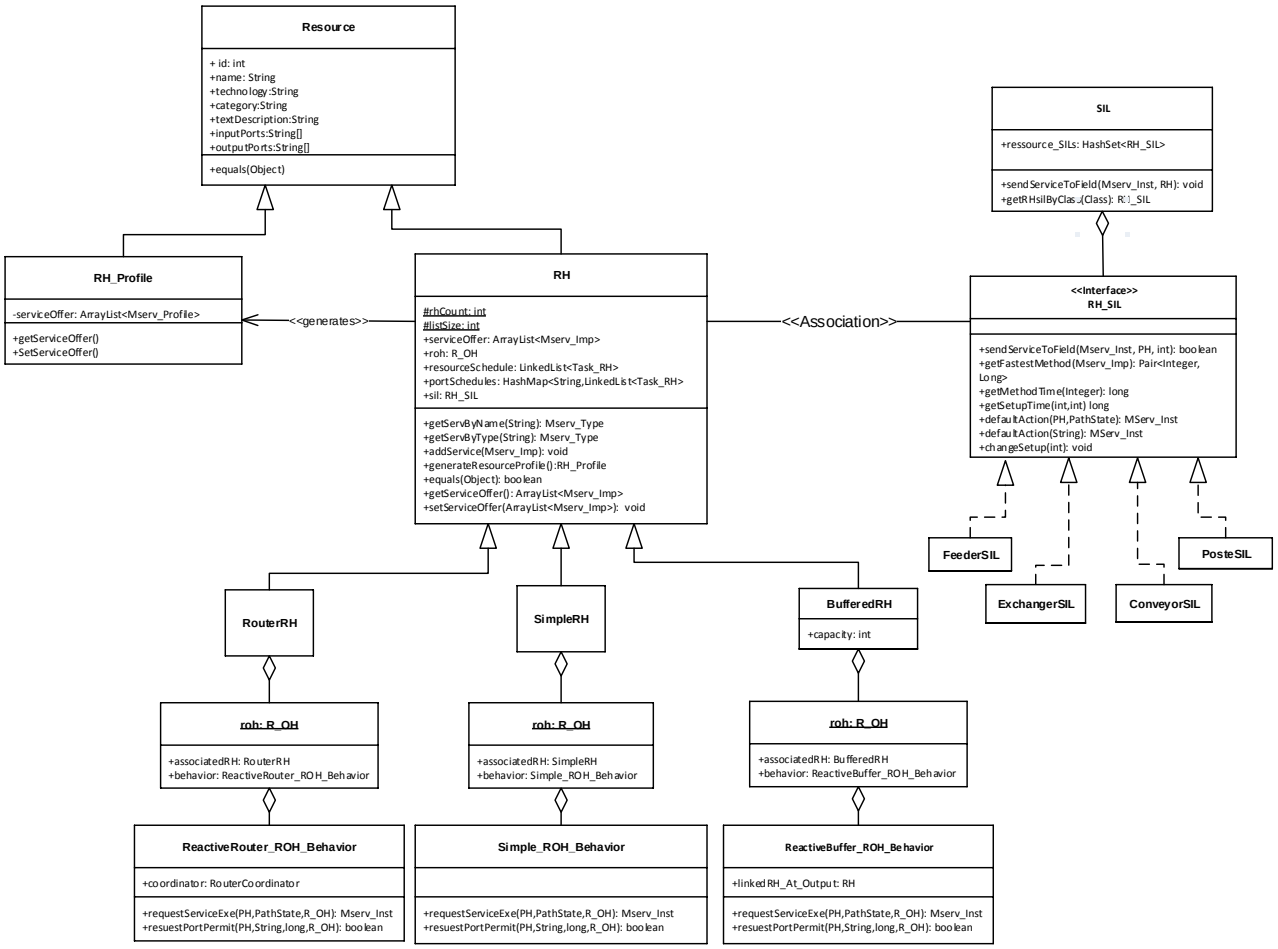


Figure 37 Diagramme de Classe UML : *Holon Ressource* et *SIL*

Dû aux différentes ressources existant naturellement, il existe le besoin de spécialiser le modèle des HR. Actuellement, nous avons identifié trois types des holons ressources (Figure 37), qui seront étudiés plus précisément lors du cas d'étude présenté dans le Chapitre 6 :

- **HR Simple** (*Simple RH*) : Il représente la forme la plus simple de HR, capable de traiter un seul produit à la fois et ayant un seul port d'entrée et de sortie.
- **HR de Routage** (*Router RH*) : Ce HR représente une ressource offrant des services de transport entre différents ports. Elle se caractérise par le fait d'avoir plusieurs ports d'entrée et plusieurs ports de sortie, mais ne peut traiter qu'un seul produit à la fois.
- **HR Tampon** *(Buffer RH)* : Ce HR représente une ressource passive avec la capacité de traiter (cumuler) plusieurs produits à la fois. Son objectif est de gérer sa capacité et l'utilisation de ses ports d'entrée.

La Figure 38 illustre également une autre décomposition possible des HR en fonction de la complexité de leur structure interne :

- **HR Atomique :** Représente une ressource indivisible complètement déterminée, i.e. dont le modèle interne peut être mathématiquement inversé. De tels holons représentent par exemple un poste de travail automatisé avec un contrôleur seul où toutes les fonctions sont programmées a priori.

- **HR Composé** (autrement appelé *HR Composite) :* Représente un groupe de machines, i.e. un groupe de HR atomiques ayant un environnement de travail partagé. La principale activité du HR composé est la coordination des opérations des différents HR atomiques. Les fonctions offertes par ce holon sont toutes des Services Composés.
- **HR Opérateur** : Représente un poste de travail manuel ou semi-automatisé nécessitant l'intervention d'un opérateur humain pour exécuter sa tâche. Ce holon publie les services que l'opérateur est capable d'offrir selon ses compétences et l'équipement disponible. La responsabilité de ce holon est la même que celle des HR atomiques en cela qu'elle réside dans l'ordonnancement au poste de travail. La différence majeure est qu'elle considère l'aspect humain dans son allocation et donc un ensemble de critères différents, tels que : fatigue, monotonie, expérience, qualité, etc.
- **HR Avatar** : Comme son nom le suggère, ce HR est un avatar d'un acteur externe au système. Son objectif principal est de mettre à disposition du reste du système des informations relatives à ces acteurs externes pour que les autres holons puissent les inclure dans leur mécanisme de prise de décision. Par exemple, un HR avatar peut être utilisé pour modéliser des ordres d'approvisionnement et des estimations de dates de livraison.

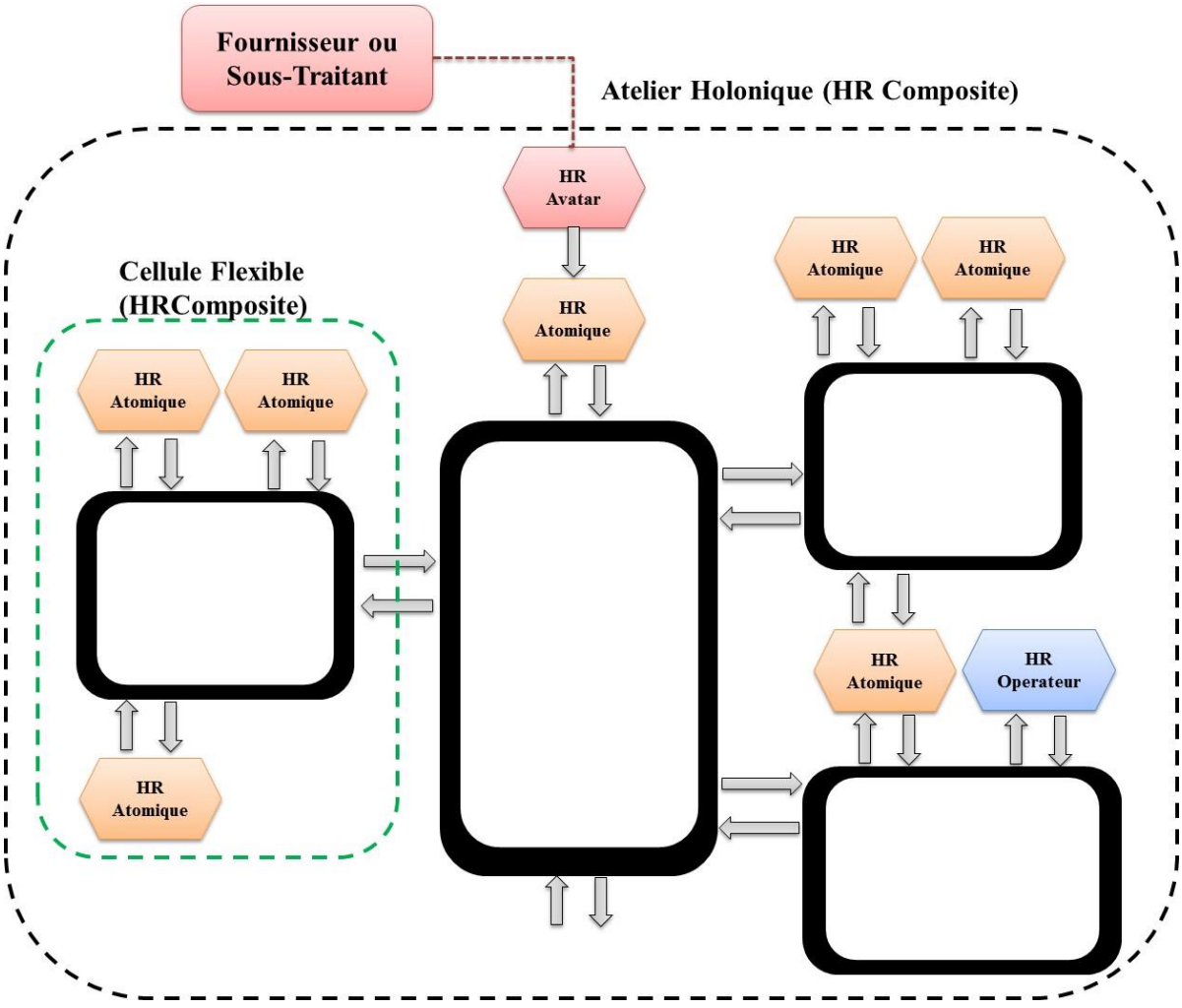


Figure 38 Représentation des quatre types de ressources

Holon Ordre

Le *Holon Ordre* est spécialisé en deux types, comme montré précédemment dans la Figure 35 , le *Holon Ordre-Produit (HOP)* et l'Holon Ordre-Ressource (HOR), chacun dédié à la coordination de l'exécution des plans de son HP ou HR associé, respectivement (Figure 39).

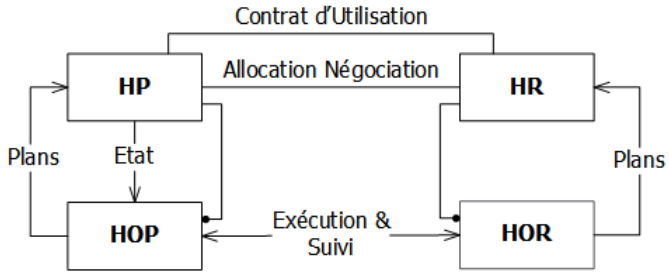


Figure 39 Diagramme de Classes UML : relations des *HOP* et *HOR*

La Figure 39 illustre les relations de ces deux spécialisations de HO. Issus des interactions de négociation entre HP et HR, des plans de production et des plans d'utilisation sont générés pour le HP et le HR respectivement. D'un côté, le *HOP* suit et coordonne l'exécution des plans de production récupérés du HP associé. Dans le cas d'une perturbation invalidant les plans, le *HOP* passe à un comportement réactif et continue la production en utilisant le modèle de gammes du HP lorsque le HP communique de nouveaux plans. D'un autre côté, le *HOR* coordonne les appels aux services gérant à ce qu'ils respectent les plans d'utilisation de la ressource.

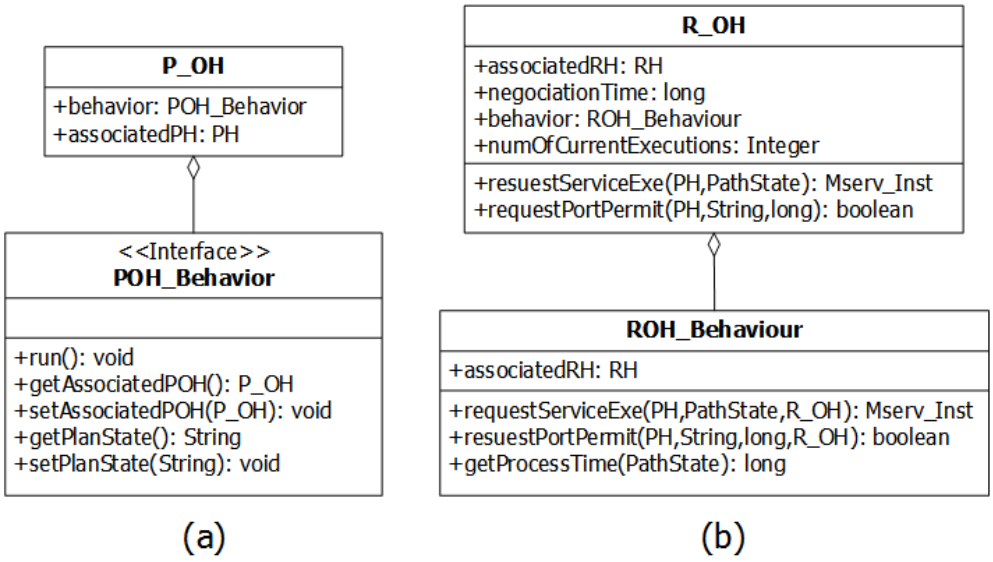


Figure 40 Diagramme de Classes UML : (a) *Holon Ordre-Produit*, (b) *Holon Ordre-Ressource*

Directory Facilitator

Le *Directory Facilitator (DF)* sert comme un serveur d'information au travers duquel les holons communiquant des informations statiques en vue de réduire le trafic des messages (et éviter une approche de communication type Broadcast). Il contient ainsi les descriptions des ressources en termes de leur offre et de leur localisation sur le système de transport, ainsi qu'une version actualisée du plan des connexions physiques de l'atelier.

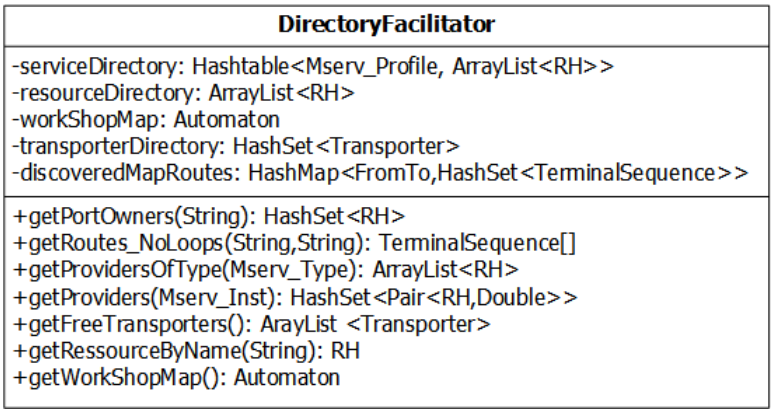


Figure 41 Facilitateur de Répertoire (*DF*)

Parmi ces services d'information, le DF peut communiquer les propriétaires d'un port [*getPortOwners()*], le calcul des routes entre deux ports dans le système de transport [*getRoutes()*], les informations pour s'adresser aux transporteurs de produit non associés [*getFreeTransporters()*], le plan de l'atelier [*getWorkShopMap()*] et les fournisseurs de services ayant une correspondance avec ses capacités.

3.5.2 Les couches abstraites

L'objet de cette section est de proposer une architecture de contrôle permettant d'insérer les concepts logiciels présentés dans les sections précédentes à une architecture matérielle descendant jusqu'au réseau de terrain et au contrôle des actionneurs (robots, bras manipulateurs, MOCN, etc.).

La Figure 42 propose une structuration de cette architecture en différentes couches. La première couche est identifiée comme étant la plateforme holonique. À l'intérieur de cette couche, nous retrouvons l'ensemble des fonctions explicitées ci-dessus, classées en fonction de leur niveau d'abstraction, i.e. leur éloignement vis-à-vis du contrôle du processus. Le plus haut niveau correspond donc à la spécification produit au niveau des HP. À partir de cette spécification, la relation HP-HO permet de générer l'orchestration flexible du processus, via une interaction avec le DF. A l'intérieur de ce DF, deux niveaux d'abstraction sont présents. Tout d'abord le nuage de services, ne contenant que les *Profils* de chaque *MService*. Enfin, le DF contient également un nuage contenant l'ensemble des parties logiques des ressources s'étant annoncées dans le système.

Cette première couche est en étroite relation avec une seconde couche ayant la particularité d'être soumise à de plus fortes contraintes temporelles. Dans cette couche, le premier niveau d'abstraction contient les HO spécialisés en *HOP* et *HOR* pour le suivi temps-réel de la production. Le lien vers la couche temps-réel strict est assuré par le *SIL*, qui permet de lier la notion de service à la méthode d'implémentation de l'opération sur le processus.

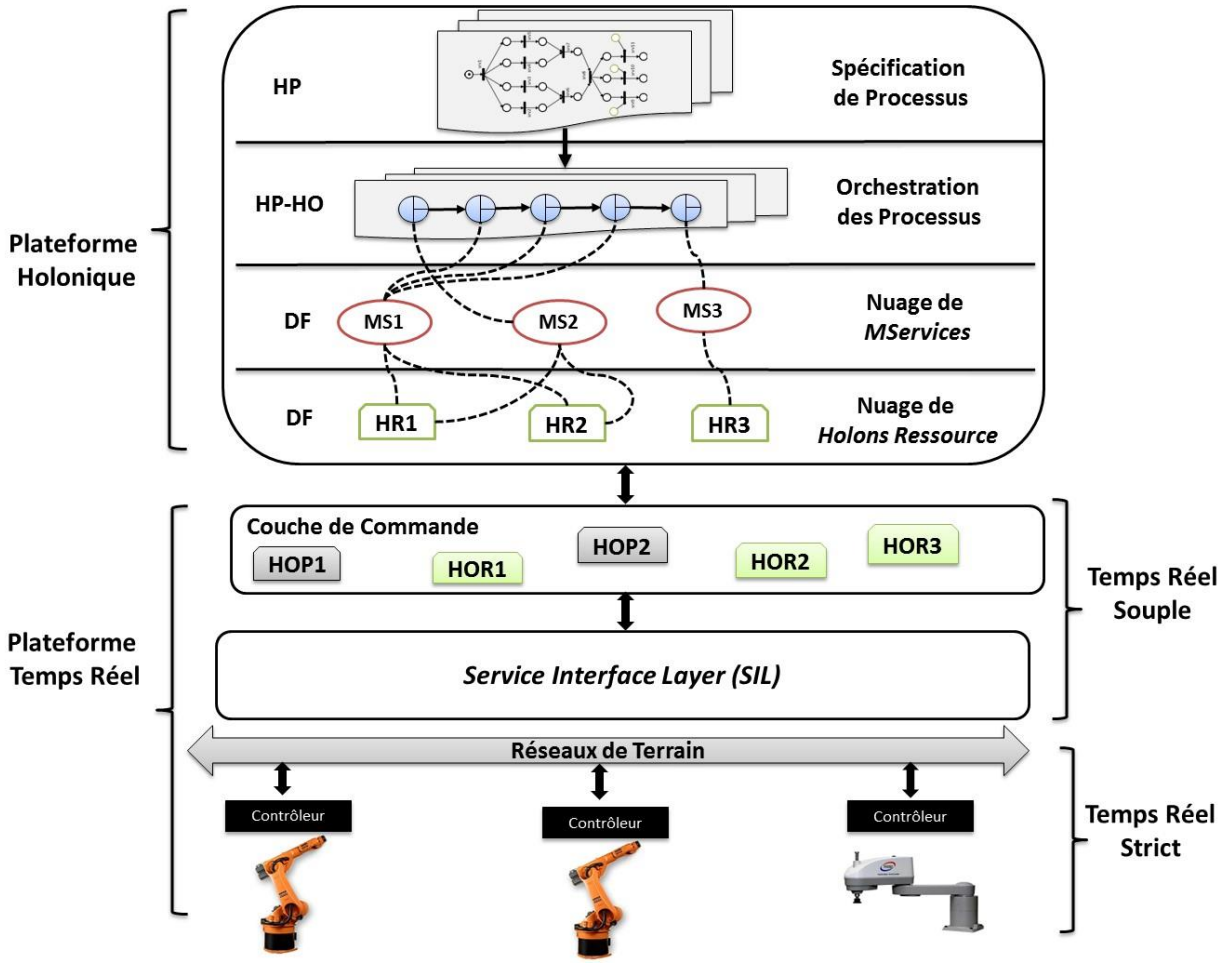


Figure 42 Architecture de commande d'un *SoHMS*

3.5.3 Création d'un nuage de services

Dans un *SoHMS*, un *MService* est créé et rendu disponible comme l'illustre la Figure 43 Tout d'abord, une *OdApp* peut être créée, donnant une description plus formelle des fonctionnalités sous forme de types de MServices. Avec une telle ontologie, fournissant des concepts et de la terminologie unifiée, on peut alors créer une description des processus compréhensible par l'ensemble des acteurs. Une telle description nécessite la définition de contraintes de relations entre les *MServices* se basant sur les descriptions des effets et conditions des *MServices* et sur des connaissances expertes des concepteurs du système. La matérialisation d'un tel service dans un nuage de *MServices* commence avec la programmation d'une méthode dans une ressource selon sa technologie et ses langages éventuellement propriétaires. Cette méthode, réalisant des transformations dans son environnement, est alors encapsulée et représentée par l'interface d'un type de *MService* et exposé au système comme tel.

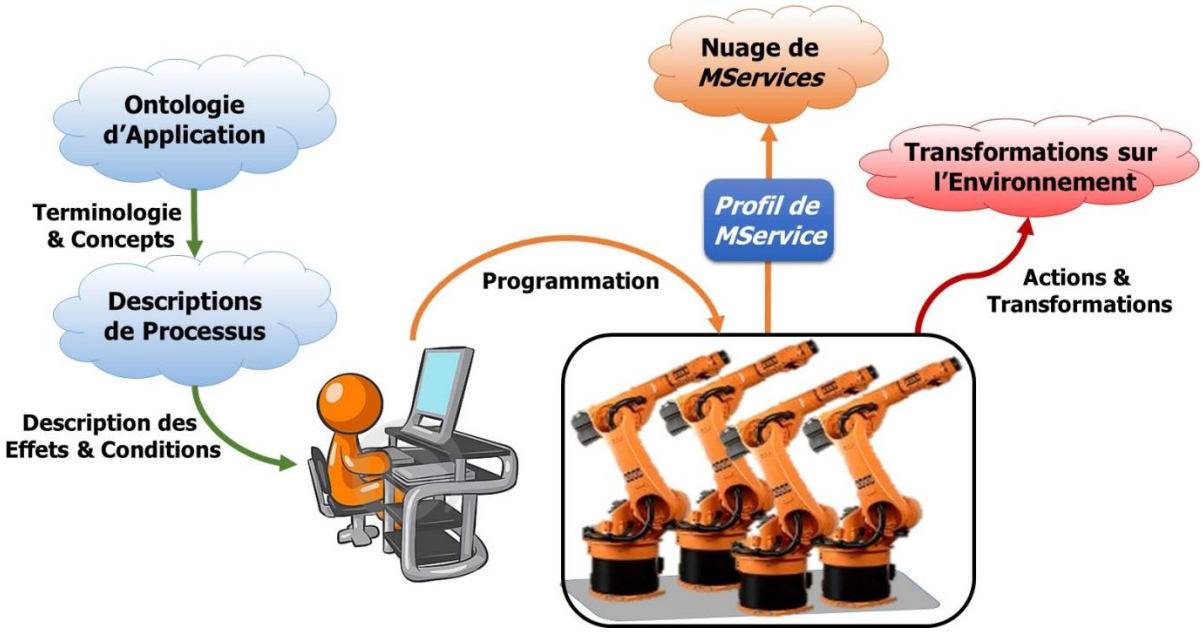


Figure 43 Création d'un Nuage de *MServices*

Les ressources, ainsi que les services, peuvent avoir une structure récursive ou fractale. De ce fait, un HR encapsulant un *SoHMS* offre une collection de services dans un nuage de services extérieurs, lesquels sont composés à partir de services provenant d'un nuage intérieur. Cette structure peut se répéter ainsi tout au long de la hiérarchie jusqu'à rencontrer des *MServices* et *HR Atomiques*.

3.6 CONCLUSION

Ce troisième chapitre avait pour objectif de présenter l'un des concepts centraux de cette thèse qu'est le *SoHMS*. La principale proposition qui est faite est la description du comportement et du fonctionnement de chaque holon vis-à-vis de l'introduction du concept de services dans le système de contrôle. La proposition se base majoritairement sur les architectures de référence présentées dans la littérature, mais de nombreuses adaptations ont été proposées.

Après avoir présenté les relations existantes entre *MServices* et systèmes holoniques, nous avons présenté en détail en quoi nous pensons que le concept de *SoHMS* peut servir de méthodologie de conception du système de contrôle. Cette méthodologie a été présentée pas à pas, et une frontière d'études des types de systèmes que nous pensons être susceptibles d'accueillir ces concepts a été définie (Figure 34). Enfin, une proposition d'architecture *SoHMS* complète, de l'interaction multi-agent au réseau de terrain, a été proposée.

Cette architecture sert de base aux différentes propositions faites dans ce document, notamment au niveau de l'orchestration des gammes flexibles présentées dans le chapitre suivant. Cette orchestration (Chapitre 5) utilisera les relations induites par les concepts développés ici. Enfin, le développement sur le cas d'études général du Chapitre 6 se basera complètement sur l'architecture développée dans ce chapitre.

Chapitre 4

Modélisation de Gammes Flexibles Orientées-Services

Dans le Chapitre 2 a été présenté le modèle conceptuel et informationnel des MServices et comment leur caractéristique récursive est utilisée pour concevoir des processus qui eux aussi peuvent être encapsulés sous la forme d'un MService. Nous avons identifié deux types de processus selon les relations de services qui les composent : Les ProcesProd-S et les ProcesDisp-S. Au niveau de la modélisation des ProcesProd-S, on constate qu'un modèle informationnel complet détaillant les informations de production d'un produit nécessite un mécanisme qui ne décrit pas seulement la spécification de ces MService mais aussi les relations entre ceux-ci : un modèle de gamme. Le concept de gamme flexible consiste à apporter de la flexibilité au système à travers cette information. Dans le contexte présenté dans le Chapitre 3, ceci consiste en la description de toutes les possibilités de production, notamment de séquencement, d'un processus en se basant sur la nature des opérations. Cela ouvre un espace de solutions plus large permettant, avec un mécanisme d'orchestration adéquat, de créer des flux de production pour tout membre d'une famille de produit.

Dans la première partie de ce chapitre, nous faisons une analyse des différentes approches et formalismes proposés dans la littérature pour la description des processus et la modélisation de gammes flexibles de production de produits. En plus des formalismes, nous proposons également une analyse des caractéristiques particulières des processus du domaine de la production. Dans la deuxième partie, nous proposons un modèle computationnel de gamme flexible basé sur une extension du formalisme de Réseaux de Petri, ainsi que deux approches pour la programmation des gammes à partir des spécifications de produit et la création d'une OdApp. Dans une troisième partie, nous présentons une série de règles de modélisation ainsi qu'un algorithme pour la génération automatique des modèles de gammes en Réseaux de Petri, exploitables par un mécanisme d'orchestration. Finalement, nous illustrons les concepts présentés dans le chapitre à travers deux cas d'études, le premier illustrant la création d'un modèle conceptuel de processus d'un produit ainsi que la construction d'une ontologie et le deuxième pour illustrer la génération du modèle computationnel de gamme flexible en se basant sur les règles de modélisation et montrer comment effectivement grâce à ce modèle de gamme flexible, indépendant de tout atelier, nous ouvrons la voie à un large espace des solutions.

Les travaux présentés dans ce chapitre ont été publiés dans une revue internationale (Gamboa Quintanilla et al., 2016), un chapitre d'ouvrage (Gamboa Quintanilla et al., 2014a) et deux conférences internationales (Gamboa Quintanilla et al., 2014b, 2013b).

4.1 INTRODUCTION AUX GAMMES FLEXIBLES DE PRODUCTION

Classiquement, les outils d'ordonnancement se basent sur une représentation de l'outil de production sous forme de capacité, disponibilité ou compétence, et des gammes de production de chaque produit, représentées par des séquences linéaires de tâches à réaliser. Ainsi, la flexibilité de l'outil, que ce soit dans le cas d'un job-shop ou d'un job-shop flexible par exemple, n'est explorée qu'au niveau des ressources, c'est-à-dire la possibilité d'affecter la même tâche à plusieurs machines alternativement.

L'objectif de cette section est d'illustrer l'utilité de disposer de flexibilité au niveau de la gamme de production, de montrer quelles sont les limites actuelles des travaux de la littérature et les possibilités que l'on peut entrevoir pour la suite des travaux.

4.1.1 Spécification produit et gammes flexibles

Traditionnellement en conception de processus, des modèles statiques sont implantés et spécifient seulement un planning de production prédéfini. En se basant sur les notions de familles de produits et de processus, il doit être possible de définir des modèles plus élaborés permettant d'augmenter la flexibilité de l'outil.

La spécification des familles de processus, tout comme les familles de produits, présente deux challenges majeurs : (i) au lieu d'être une collection de processus individuels, la spécification des processus devrait expliquer les relations entre les processus, (ii) un processus individuel devrait être défini a priori de la sélection des paramètres liés à la famille de produits, où de tels paramètres sont le résultat des spécifications clients. De ce fait, la description d'une instance spécifique de processus est une fonction liant à la fois : les paramètres de spécification et une description de la famille de produits, comme décrit dans (Jiao et al., 1998) au sujet des familles de produits. La réalisation d'une famille de processus peut donc être offerte en tant que *MService* avec un ensemble de paramètres de service et une description des caractéristiques du produit en tant qu'effet. Etant donné que la structure du processus dépend des méthodes et de la technologie utilisée par les ressources qui produisent le service, cette information est fournie par la méthode du processus dans le modèle d'implantation du *MService* et son propriétaire est le *Holon Ressource.*

Dans le contexte de la conception d'une famille de produits personnalisables, il y a deux challenges principaux dans l'organisation du modèle de données. Premièrement, au lieu d'être une collection de variantes de produits individuels, le modèle devrait expliciter les relations entre ces variantes. Deuxièmement, une variante individuelle devrait être définie a priori de la sélection des paramètres liés à la famille de produits. De tels paramètres sont le résultat de la spécification du processus par le client, i.e. la personnalisation du produit. La différentiation des produits est alors effectuée par, à la fois, la spécification des services et la configuration des différents services de production au travers de l'ajout, du retrait et/ou de la substitution de ceux-ci. De ce fait, une description spécifique du processus de production d'une variante de produit est une fonction à la fois de la spécification des paramètres et d'une description de la famille de processus.

Une spécification produit doit exprimer toutes les informations nécessaires à la production d'un produit spécifique : paramètres, services de production et conditions de précédence des services. Son outil de modélisation doit être capable d'exprimer toutes les chorégraphies de services possibles qui peuvent produire le produit spécifié. De plus, elle doit également faciliter l'édition en ligne, être facile à comprendre et à programmer et doit avoir une faible consommation mémoire pour faciliter une potentielle application embarquée.

4.1.2 Besoin d'un modèle computationnel de gamme

Au niveau produit, l'objectif est d'avoir un mécanisme d'orchestration (i.e. modèle et moteur) à la fois flexible et réactif : flexible dans sa capacité à donner des solutions différentes au cours du cycle de vie du produit et réactif pour donner des solutions valables (pas nécessairement la meilleure) au fur et à mesure de la production et à des moments pertinents considérant les états actuels du produit et de la production. La construction de ce modèle nécessite la prise en compte de 5 types de contraintes (Henrioud and Bourjault, 1991) :

- *Contraintes Géométriques* : dépendent de la structure du produit et prennent en compte toutes les relations entre sous-éléments telles que parent-enfant ou voisinage par exemple ;
- *Contraintes d'Accessibilité :* permettent d'éviter les situations de blocages pour cause d'interférences de position ou de trajectoire ;
- *Contraintes de Stabilité* : chaque sous-assemblage doit être stable, i.e. l'état du produit ne peut changer dans le futur sans l'exécution d'un autre service ;
- *Contraintes de Production* : prennent en compte toutes les autres contraintes liées aux processus de production.

Le modèle de *SOP* présenté dans le Chapitre 2 définit les éléments informationnels nécessaires pour complètement spécifier la gamme. Cette gamme doit suivre un modèle exploitable à la fois par les services de planning et d'ordonnancement du système. Un tel modèle se doit d'être computationnel et doit :

- Contenir toutes les informations de processus nécessaires à la réalisation d'une variante de produit ;
- Décrire la structure générale d'une famille de processus contenant toutes les informations sur le processus et les possibles ramifications en termes de séquencement ;
- Accueillir les paramètres d'échelle et modulaires définis lors de la personnalisation ;
- Agréger toutes les différentes alternatives de production sous un seul format ;
- Fournir un haut degré de flexibilité de reconfiguration ;
- Suivre un format intelligible pour l'utilisateur ;
- Etre facile à comprendre et programmer sans avoir besoin de connaissances spécifiques en informatique afin de les rendre accessibles pour le concepteur des processus ;
- Etre compatible avec une évaluation en ligne suffisamment rapide pour fournir des solutions valides en conditions temps-réel pour augmenter la réactivité du système.

En résumé, le système de pilotage a besoin d'un modèle informatisé qui peut représenter toutes les séquences de production possibles réalisant différentes variantes de produits, dans un format unique et simple à lire, que les planificateurs peuvent facilement manipuler et concevoir et permettant une exploration rapide des possibilités. De plus, le modèle doit être indépendant de l'atelier considéré. L'idée est de définir ce modèle selon une *OdApp* et d'utiliser le système de pilotage pour identifier et explorer les capacités de l'atelier pour générer les plans de production de manière dynamique. La seule contrainte pour qu'un modèle de gamme puisse être déployé sur un système est donc qu'à la fois le modèle et le système soient bases sur la même ontologie de services.

4.1.3 Etat de l'art : formalismes de modélisation

L'objectif de cette section est d'étudier plusieurs formalismes de modélisation de la littérature afin de choisir le mieux approprié par rapport aux contraintes énoncées ci-dessus.

Le *« Process Specification Language »* (PSL) (National Institute of Standards and Technology (NIST), 2008; Schlenoff et al., 1999), développé au NIST (National Institute of Specification Technology)[1], est une proposition d'ontologie fournissant une description formelle des composants et relations formant un processus. Il fournit un ensemble de termes logiques pour la description de processus et un vocabulaire de classes et de relations tel que: *Activité*, *Occurrence d'Activité*, *Point de Temps*, *Object*, *Fluent*, *Arbre des Occurrences*, *Cours de Evènements*, etc. Son objectif principal est d'axiomatiser un ensemble de primitives sémantiques intuitives, adéquates à décrire les concepts fondamentaux régissant les processus de production, avec pour objectif de faciliter des échanges d'information complets et corrects entre applications de systèmes de production. Cependant, une perspective prédécesseur-successeur est utilisée pour la spécification des relations de précédence. Cette perspective n'est probablement pas la meilleure pour spécifier des relations complexes et multiples. En effet, il est beaucoup plus facile et confortable de réfléchir en termes de préconditions nécessaires à être validées pour qu'un service puisse être exécuté plutôt que de réfléchir à l'ensemble des successeurs possibles à ce service. De plus, ce langage ne considère pas le concept de services. Toutefois, beaucoup de concepts, terminologies et orientations de PSL peuvent être réutilisés dans la perspective mise en œuvre dans ce document.

Au sein des technologies de services web, le *Web services Description Language* (WSDL) est un langage basé sur XML dédié à la description abstraite d'une interface de service pour sa propre invocation. Une telle description d'interface contient des informations à propos d'une tâche spécifique. Cependant, elle ne spécifie pas les relations (séquentielles ou temporelles) qui peuvent exister avec d'autres services de manière à former des processus plus complexes. Le *Business Process Exécution Language* (WS-BPEL), développé par OASIS[2] et également basé sur XML, est un langage utilisé pour décrire les différents workflows pouvant être composés par l'expression des motifs d'interaction de la collection de services web trouvés dans un processus. Malheureusement, comme indiqué dans (Jammes and Smit, 2005), la plupart de l'offre de moteurs d'orchestration utilisant ces langages et méthodes ne sont pas utilisables pour décrire des processus industriels. La raison principale est la verbosité des langages basés sur XML, qui implique des tailles de fichiers de l'ordre de 10MB. Cet inconvénient ne se ressent pas pour des applications au niveau entreprise, mais il est indésirable pour les systèmes industriels actuels ou à venir (basés sur des systèmes cyber-physiques (Colombo et al., 2014), produits intelligents ou actifs (Sallez, 2014) par exemple), où la coordination des services et les moteurs d'orchestration ont vocation à être embarqués dans des systèmes plus ou moins mobiles ayant des capacités de stockage et de traitement limitées.

Un nouveau paradigme a émergé avec l'apparition des technologies de services web sémantiques, qui a pour objectif de définir comment des composants intelligents représentent et traitent les informations à propos d'eux-mêmes, d'autres composants ou de l'environnement (Delamer and Lastra, 2006). L'idée principale est de fournir une sémantique interprétable par les machines afin que les équipements puissent acquérir de la connaissance à propos des autres équipements au travers d'inférence et de traitements automatisés plutôt qu'une correspondance directe d'ensembles de compétences. OWL-S, développé par le W3C[3], est un langage ontologique web sémantique fournissant le vocabulaire standard pouvant être utilisé pour créer des descriptions interprétables automatiquement pour accéder aux services proposés. Ce langage ontologique fournit un framework complet pour définir les modèles de services sémantiques.

[1] http://www.mel.nist.gov/psl/
[2] https://www.oasis-open.org/committees/wsbpel/
[3] http://www.w3.org/Submission/OWL-S/

Bien que cet axe soit très prometteur pour améliorer l'intégration des systèmes intelligents, son applicabilité à court terme peut être discutée du fait de sa complexité et du grand effort de préparation qu'il demande aux programmeurs et concepteurs de processus, ce qui serait dommageable au niveau de la réactivité en phase de conception. De plus, comme peut l'être indiqué dans (Jammes et al., 2005) par exemple, cette approche est principalement centrée sur la composition de services à un bas niveau, au niveau de l'équipement, ce qui est un niveau plus bas que celui considéré dans ce document.

Toujours en phase de conception, les modèles graphiques ont la particularité d'être les plus faciles à comprendre, concevoir et éditer par rapport aux modèles présentés précédemment. IDEF3 par exemple est une spécification de processus basée sur un langage graphique permettant de décrire les processus et représenter sa structure. La description de processus IDEF3 créé un réseau de relations entre activité dans un scénario donné avec des éléments tels que les Unités De Comportement, liens ou jonctions pour former des schémas tels que celui présenté Figure 44 . Bien que ce langage graphique soit principalement destiné à la représentation à destination d'opérateurs humains, il est concevable d'imaginer un outil qui viendrait automatiquement commander et surveiller les process à partir d'une représentation IDEF3. Toutefois, il n'existe pas d'outils d'analyse des arrangements graphiques d'IDEF3 comme il en existe pour d'autres méthodes graphiques.

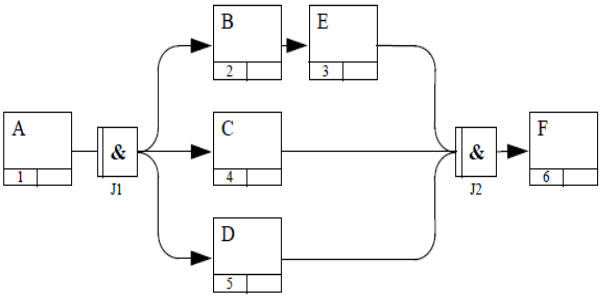


Figure 44 Exemple de représentation IDEF3

Le formalisme *Réseaux de Petri* (RdP), bien connu à la fois dans les univers académiques et industriels, se trouve être un très bon moyen pour construire un tel modèle d'orchestration et fournir le mécanisme logique pour l'orchestration des séquences de production grâce à sa capacité à capturer les relations synchrones et asynchrones entre les tâches, à la facilité de prise en main de son langage graphique et de son empreinte minimale en mémoire qui le rend très intéressant pour les applications embarquées telles que celles sur des entités mobiles (Mendes et al., 2010). En détail, la structure du réseau permet de générer un grand nombre de combinaisons de séquences lui permettent de minimiser son empreinte en mémoire, en opposition à WS-BPEL par exemple. Ensuite, le réseau est basé sur les états, ce qui est très avantageux pour la prise de décision en temps réel, puisque des calculs peuvent être lancés et relancés à tout instant basé sur l'état actuel ou futur du réseau. Enfin, le formalisme est très répandu au sein des concepteurs de processus et programmeurs, avec de fortes fondations mathématiques et de nombreux outils d'analyse disponibles. En conséquence, ce langage graphique de modélisation peut être utilisé pour représenter des familles de processus de manière compacte et sa logique donnera naturellement de la flexibilité aux moteurs d'orchestration niveau produit en découlant.

4.1.4 Etat de l'art : Gammes Flexibles

De récents travaux sur les HMS proposent d'enrichir le modèle de spécification de processus en considérant l'existence de différentes opérations alternatives à réaliser à un état donné, comme dans (Mendes et al., 2009) et (Mendes et al., 2010) où les auteurs proposent l'utilisation de contrôleurs basés sur des RdP pour modéliser les alternatives. De même, (Toguyeni, 2006) et (Bourdeaud'huy and Toguyeni, 2006) suggèrent la création d'une *Logical Operating Sequence* représentant la structure du

processus mais indépendante de l'atelier considéré. Toutefois, aucun de ces travaux ne se penchent sur la manière de créer ces modèles ou de décrire les structures de processus. Cette tâche est laissée au concepteur de processus afin qu'ils puissent mettre à profit leur expertise sur le processus et leur expérience de manière générale. De plus, les opérations formant un processus sont généralement conçues et programmées pour un équipement spécifique, sans souci de réutilisation ultérieure sur d'autres équipements ou dans d'autres processus. Cette section a pour objectif de lister les approches de la littérature sur de tels modèles de spécification de processus.

La génération de gammes non-linéaires (*Non-Linear Process Plans*, NLPP) a été automatisée par les systèmes CAPP (*Computer Automated Process Planning*). A partir des informations liées au produit concernant ses caractéristiques et les relations entre ceux-ci, il s'agit de créer des modèles de gammes représentant des alternatives de production. Les outils de modélisation utilisés sont des graphes ET/OU et des RdP. Le projet FLEXPLAN (Kruth and Detand, 1992) est un exemple d'approche NLPP. Dans cette approche, un système CAPP générant automatiquement des NLPP est implanté utilisant 3 relations basiques : composition, direction de fabrication différente ou identique. Dans le processus, deux représentations sont créées : premièrement, un model générique en RdP est construit et contient toutes les relations géométriques entre caractéristiques (qui peuvent être éditées pour ajouter des contraintes non-géométriques requérant une expertise humaine) ; deuxièmement, un graphe ET/OU est créé reliant les opérations aux ressources. De manière à réduire l'explosion combinatoire, FLEXPLAN utilise une méthode hiérarchique pour créer des groupes de caractéristiques, donc des opérations. Cette approche de génération de modèle présente cependant l'inconvénient de ne pas garantir, lors de l'évolution du RdP utilisé pour générer le graphe ET/OU, que le prochain pas sera une solution faisable. Une simulation itérative doit être effectuée pour identifier les séquences valides d'opérations, contribuant au problème d'explosion combinatoire. Un autre inconvénient est la perte d'une partie de la flexibilité liée aux ressources en contraignant toutes les exécutions d'opérations consécutives à une machine. Le projet COMPLAN (Kruth et al., 1996), faisant également partie du projet européen ESPRIT[4], essaie d'appliquer les principes de FLEXPLAN sur la génération de gammes contenant des alternatives pour des lots de petites tailles de produits mécaniques complexes dans un environnement job-shop typique. La contribution majeure du projet a été la définition de méthodes pour améliorer le temps de réponse du système CAPP qui génère les NLPP, qui peuvent être reprises dans les stratégies de génération présentées dans le reste de ce chapitre. Une approche de génération de gammes d'assemblage assistée par ordinateur est également présentée dans (Henrioud and Bourjault, 1992). Le principe est de représenter la structure du processus avec des arbres d'assemblage et un graphe de précédence avec des relations binaires entre opérations prises deux à deux. Un hypergraphe doit ensuite être créé, combinant les deux graphes, créant des relations binaires entre une opération et une liste d'autres opérations. Des avancées ont aussi été proposées pour la réduction de la complexité combinatoire avec l'insertion de contraintes stratégiques, telles que des sous-assemblages imposés, groupements d'opérations ou des arbres d'assemblages linéaires.

Une critique classique de ce type d'approche est que la génération automatisée a une approche statique et hors ligne, partant des fonctionnalités offertes par les outils de CAO (Conceptions Assistée par Ordinateur), et oublie généralement les effets bénéfiques potentiels d'une intégration descendante venant des fonctionnalités d'ordonnancement (Tan and Khoshnevis, 2000). Cette intégration se limite classiquement à la définition de règles locales d'ordonnancement et vient dans un deuxième temps, après la définition des graphes NLPP. De plus, peu d'attention est portée aux effets des différents aléas en cours de production sur la performance du système.

[4] http://cordis.europa.eu/programme/rcn/197_en.html

4.1.5 Choix de l'outil de modélisation

Le formalisme RdP a été choisi pour la suite de nos travaux, car nous pensons qu'il est le meilleur candidat pour la conception de modèles *SOP* informatisés. Les RdP sont bien connus dans les deux mondes académique et industriel, ce qui en fait un outil compréhensible pour les ingénieurs finalement en charge de leur utilisation. Ses fondements théoriques forts, les techniques d'analyse puissantes et abondantes et sa capacité à figurer des relations synchrones et asynchrones entre les transitions dans un format compact sont autant d'atouts qui soulignent sa pertinence. Grâce à ses primitives de base et ses capacités d'évolution, un RdP seul a la capacité de modéliser tout type de dépendances complexes et générer un grand nombre de combinaisons de séquences avec une empreinte minimale de mémoire, ce qui est intéressant pour les applications embarquées.

4.2 MODELE DE GAMMES FLEXIBLES EN RESEAUX DE PETRI

Le choix a donc été fait de modéliser les processus sous forme de gammes flexibles avec l'outil RdP. Il reste dans cette section à définir la représentation sémantique des éléments du processus définis dans le modèle du produit, définir les règles de modélisation nécessaires à la création d'un réseau conforme à la structure du produit et la dynamique de l'orchestration au niveau produit utilisant ces réseaux.

4.2.1 Présentation du modèle

Ce réseau est conçu pour exprimer de manière compacte toutes les relations de précédence pouvant être exprimé dans le processus de production. Les places représentent des permissions pour exécuter des services, représentés par les transitions. L'état du produit dans son cycle de production est donc indiqué par son marquage. Grâce à une extension du formalisme de RdP (Murata, 1989), des arcs inhibiteurs évitent au réseau de se retrouver dans des situations de verrou.

D'autres formalismes RdP de haut niveau tels que les RdP colorés (Colored Petri-Nets – CPN) sont actuellement la cible de nombreux projets de recherche. Toutefois, le formalisme RdP classiques avec l'extension des arcs inhibiteurs (Roux and Lime, 2004) semble être une solution suffisante pour la modélisation de processus au niveau produit. Plusieurs raisons justifient ce choix:

- La méthode est conçue suivant l'hypothèse selon laquelle les effets des services sont déterministes, i.e. le résultat de l'exécution d'un service donne toujours le même résultat. Ceci élimine l'intérêt principal des CPN qu'est le branchement conditionnel ;
- La lecture et la compréhension des réseaux classiques est plus graphique et plus aisée que les CPN ;
- Les CPN contiennent plus d'éléments dans leur définition, ce qui augmente leur empreinte en mémoire ;
- Le catalogue d'outils d'analyse est plus restreint pour les CPN ;
- L'ajout des arcs est suffisant pour créer des modèles simples à comprendre mais capables de représenter toutes les séquences possibles dans un processus de type job-shop flexible.

4.2.2 Le formalisme

Le modèle de RdP est décrit par le 5-tuple: $PN'= (P,T,A,W,Inh,Read,FI,M0)$ où :

- $P=\{p1,p2,...,pn\}$ est un ensemble fini de places. Il y a trois types: *Places de Permission*, *Places d'Exclusion Locale* LE( ), et *Places d'Exclusion Mutuelle* ME( ). Les places de permission enregistrent l'exécution de tous les services ayant un minterme (ET logique) dans leur

condition de précédence. Ce sont les leviers principaux d'exécution. Les places d'Exclusion Locale sont utilisées pour enregistrer l'exécution d'un service avec un maxterme (OU logique) ou un opérateur de négation dans sa condition de précédence. Les places d'exclusion mutuelles sont utilisées pour éviter des situations de verrou.

- $T=\{t1,t2,\ldots,tm\}$ est un ensemble fini de transitions. Chaque transition, sauf les transitions Start et Finish, est associée à un service: $S=\{s1,s2,..,sn\}\rightarrow T=\{t1,t2,..,tn\}$, S étant l'ensemble des services formant le modèle de processus.
- $A \subseteq (P\times T)\cup (T\times P)$ représente l'ensemble fini d'arcs des places aux transitions et des transitions aux places. Ils capturent toutes les relations de précédence déclarées dans la table de précédence.
- $W:A\rightarrow\{k\}$ est la fonction de pondération des arcs, avec k le poids des arcs. Dans un arc $(P\times T)$, le poids représente le nombre d'éléments capturés par l'entrée de la place de Permission, i.e. le nombre d'éléments dans la condition minterme.
- $M0$ est le marquage initial du réseau, ce qui correspond à un jeton dans chaque place ME() et LE( ) et la place P(Init), le reste des places restantes vides
- $Inh:(P\times T)*W\rightarrow (N\setminus\{0\})\cup\{+\infty\})$ est la fonction d'inhibition. Les arcs inhibiteurs, aussi appelés arcs zero-test, servent à limiter l'exécution de certains services qui pourraient amener le réseau dans une situation de blocage. Ils testent si une place dans le réseau n'a pas de jeton.
- $Read:(P\times T)*W\rightarrow N$ est la fonction de lecture. Les arcs de lecture sont utilisés pour tester la présence de jetons dans les places. Elle est utilisée pour réduire le nombre de places dans le réseau et le rendre plus facile à lire.

Dans ce formalisme de RdP étendu, une transition t est validée si et seulement si :

$$Inh(\bullet,t) \leq (M\geq Pre(\bullet,t)+Read(\bullet,t)$$

De ce fait, la fonction d'état de transition $f:Nn\times T\rightarrow Nn$ reste la même que dans le formalisme classique de RdP:

$$M'=M-Pre(\bullet,t)+Post(t,\bullet),\ soit\ \forall p \in P\ M(p)'=M(p)-Pre(p,t)+Post(t,p)$$

où $Pre(\bullet,t)=(P\times T)^*\ W\rightarrow N$ est la relation de précondition et $Post(t,\bullet)=(T\times P)^*\ W\rightarrow N$ est la relation de post-condition.

Il est important de noter que les places ne représentent pas l'état des processus comme il est classiquement d'usage dans la modélisation en RdP. L'état de la production est représenté par le marquage du réseau $M=[M(p1), M(p2),\ldots M(pn)]$ indiquant les valeurs des places de permission et d'exécution, ce qui exprime quels services ont déjà été exécutés.

Toutes les transitions sont en parallélisme structurel, exceptées celles liées à des places LE() ou ME() qui sont en conflit structurel. Toutes les transitions qui sont en parallèle sont permutables. Un marquage M est considéré comme un état de décision si et seulement si $\neg (t1,t2 \in T), t1\neq t2$ pour lequel $Pre(\bullet,t1) \leq M \geq Post(\bullet,t2)$. Ceci signifie qu'un marquage représente un état de décision s'il permet le tir de plus d'une transition pour lesquelles le moteur d'orchestration doit faire un choix sur laquelle exécuter en suivant dans la séquence.

4.2.3 Discussion sur les propriétés du réseau obtenu

Le réseau présenté ici est acyclique, où l'état de blocage représente la fin du processus de production. Aucun service n'est une transition vivace, mais tous sont des transitions quasi-vivaces. Ceci signifie que chaque service est exécuté exactement une fois durant la production. Les places de permission sont limitées dans le nombre d'éléments contenus dans la condition minterme qu'elles représentent.

Comme mentionné précédemment, le modèle est capable de générer toutes les séquences potentiellement faisables du processus. Toutefois, le nombre de ces séquences augmente très rapidement avec le nombre de services considérés, ce qui est généralement prohibitif pour des produits ayant plus d'une dizaine de composants (Henrioud and Bourjault, 1991). Il est difficile de discuter de ce point d'un point de vue purement théorique car le nombre de solutions dépend non seulement du nombre de services considérés, mais également du nombre et du type de relations de précédence. De plus, la taille de l'espace des solutions dépend également du niveau avec lequel l'ontologie de services d'application est définie. L'agrégation d'opérations de production en services de plus haut niveau peut aider à la réduction de l'explosion combinatoire, mais en sacrifiant bien évidemment un peu de flexibilité. Il est du ressort du concepteur de l'ontologie de services d'identifier le juste niveau de granularité à appliquer.

4.3 METHODOLOGIE DE MODELISATION

Il y a deux approches possibles pour définir les informations de processus et générer le modèle *SOP* en RdP : une *Approche Orientée-Services* et une *Approche Orientée-Produit,* Figure 45 . Dans chaque approche, la première étape est la définition de structures de produit ou de familles de produit. Cette structure, étant une collection de caractéristiques produit interconnectées, peut être identifiée à une collection de services de production interconnectés. Cette situation est basée sur le fait que les similitudes des structures de produit se traduisent généralement en similitudes d'opérations (Martinez et al., 2000; Schierholt, 1999), ce qui permet finalement la naissance de famille de processus. Une telle identification est possible grâce aux connaissances de production telles que celles utilisées par les systèmes CAPP (Kruth et al., 1996; Kruth and Detand, 1992). Le résultat d'une telle identification peut soit être une liste de services de production ou une liste de modèles de RdP élémentaires associés à chaque caractéristique du produit.

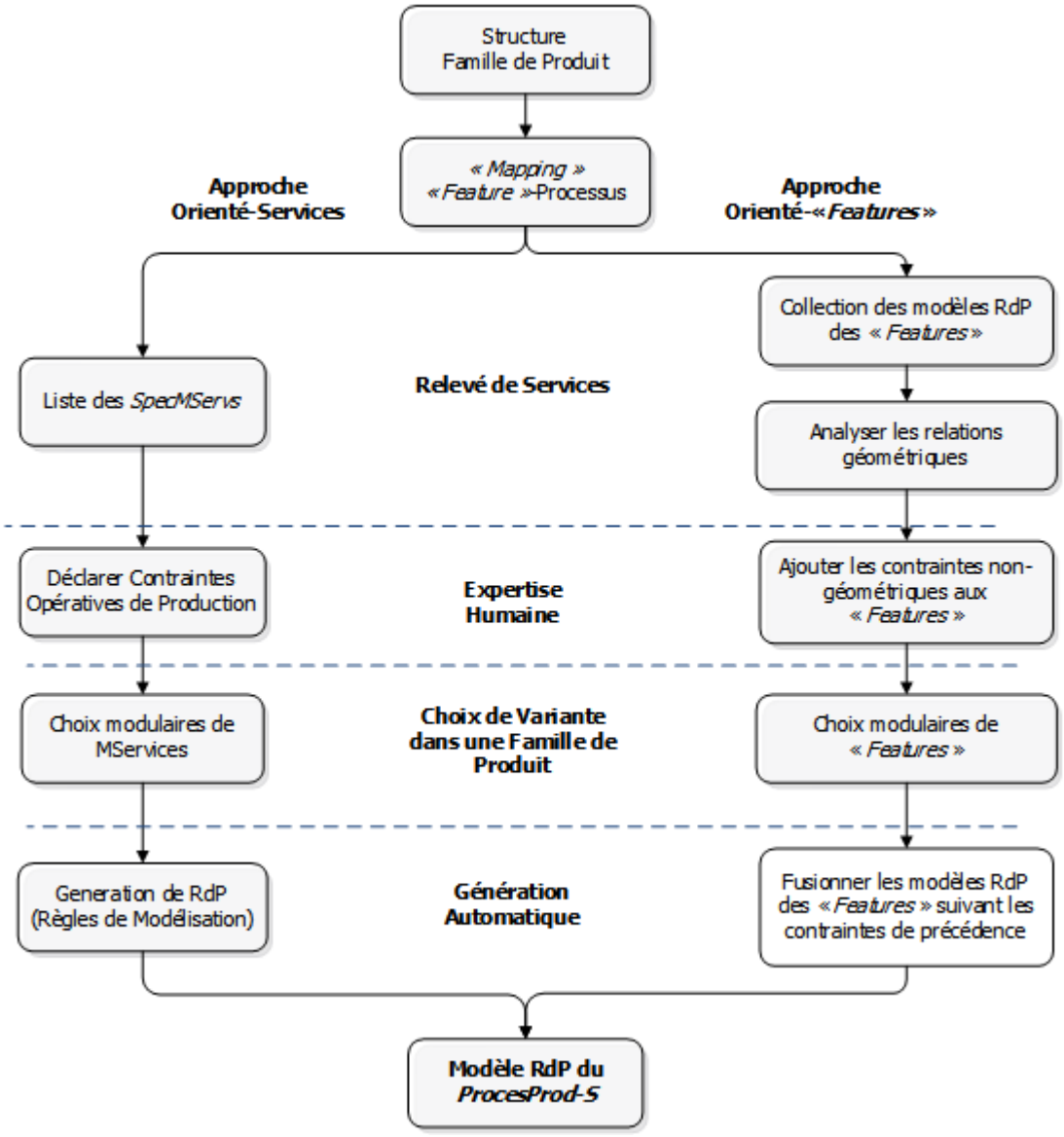


Figure 45 Processus de conception de modèles *ProcesProd-S*

Dans cette approche, il suffit donc de générer une liste de services avec leurs conditions de précédence à partir de la géométrie du produit. A partir de cette table, des contraintes non-géométriques peuvent être ajoutées par l'expertise humaine. Un ensemble de règles de modélisation va ensuite permettre de générer les réseaux élémentaires, puis le modèle *SOP* global, Section 4.4. Grâce à ce modèle global, le moteur de planification sera apte à créer un graphe d'atteignabilité représentant l'ensemble de l'espace de solutions formé par toutes les séquences de services faisables générées par le modèle. Une séquence est dite faisable si la séquence de services respecte les contraintes de précédence définies précédemment. Ces séquences n'ont toutefois aucune relation avec l'atelier, car il n'est pas encore défini comment les services seront exécutés sur le système. La tâche d'identifier, allouer et ordonnancer les séquences sur l'atelier pour créer des gammes est laissée au système de contrôle holonique, ce qui sera présenté plus en détail dans le Chapitre 5.

4.3.1 Définition de structure de produit

La Figure 46 décrit les étapes nécessaires pour aller de la définition des besoins client et exigences fonctionnelles jusqu'à la définition d'un modèle de famille de processus prêt à être utilisé lors de la spécification des paramètres de personnalisation. L'application de ces étapes pour l'ensemble des produits déclarés dans le système amènera à la constitution d'une bibliothèque de services de production, basée soit sur la définition d'une ontologie spécifique à l'application, soit sur une ontologie unifiée du domaine telle que celle présentée par la norme allemande DIN 8593 (Deutsches Institute für Normung e. V., 2003)[5] pour les processus d'assemblages.

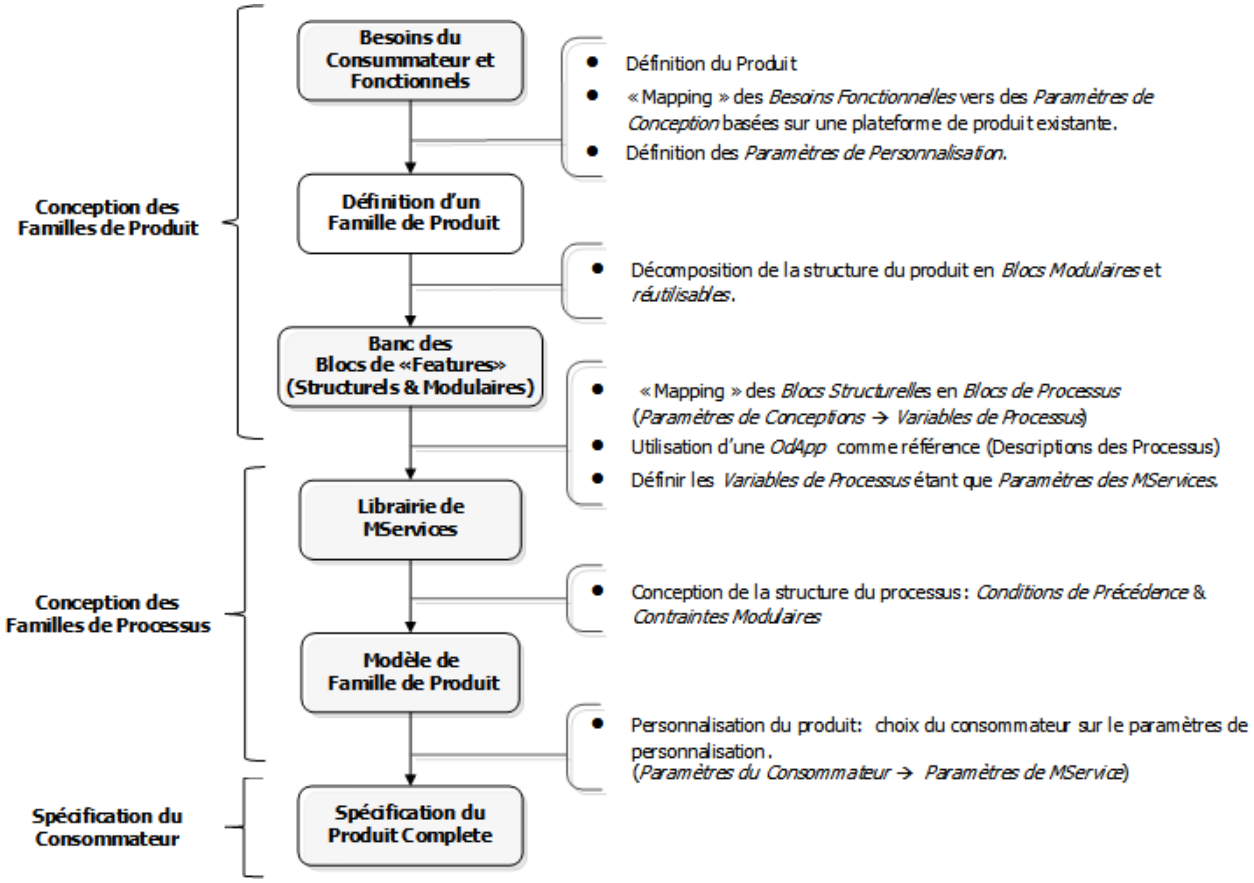


Figure 46 Framework de la conception produit jusqu'à la spécification des processus

[5] Deutsches Institute für Normung e. V.: Manufacturing Processes Joining, (2003)

4.3.2 Définition d'une *Table de Précédences*

Le modèle d'orchestration est d'abord construit à travers un tableau de priorité qui est défini par les concepteurs de processus avec une perspective de prédécesseur. En effet, il est beaucoup plus facile et moins épuisant de penser à ce que sont les conditions préalables en termes de prédécesseurs pour qu'un service soit exécutable. Dans ce tableau, les concepteurs de processus établissent, pour chaque service, une condition préalable basée sur l'exécution des services en utilisant des constructions logiques.

Cela simplifie le problème de conception, car cela évite la tâche complexe de décrire l'état d'un produit et l'espace de travail du poste en utilisant une ontologie de domaine unifiée qui est généralement inexistante. En outre, dans de nombreux cas, la définition des conditions préalables et des effets d'un service doivent être basés sur la connaissance de tout le cycle de vie d'un produit. Ceux-ci ne peuvent pas être définis indépendamment comme une propriété fixe du service. Ainsi donc, comme les conditions préalables sont établies en fonction de la connaissance du cycle de vie de produit, il est plus facile et plus simple de définir ceux-ci en termes de services plutôt qu'en termes d'états du produit.

Cette table de précédence contient la structure complète du processus en associant à chaque instance de service un ensemble d'instances de services qui doivent précéder son exécution. De telles conditions de précédence sont écrites en utilisant des opérateurs booléens (ET, OU, NOT) avec une perspective de prédécesseurs. D'autres formalismes de spécifications (tel que PSL par exemple) utilisent une perspective de successeurs, ce qui ne semble pas très pratique dans le cas de la génération d'un modèle *SOP*, car le concepteur de processus aurait à définir l'ensemble des successeurs possibles à un état de production donné, ce qui peut être un ensemble très grand selon la flexibilité du processus. Avec une perspective de prédécesseurs, cette tâche est dévolue au moteur d'orchestration.

Une première version de la table de précédence peut généralement être proposée automatiquement à partir des seuls caractéristiques géométriques des produits ou de l'outil de production. Puis, l'expertise humaine peut intervenir pour ajouter de nouvelles contraintes plus difficiles à modéliser de manière automatique. Enfin, pour distinguer les variantes de produits, un ensemble de choix modulaires peuvent être ajoutés, soit facultatifs (une caractéristique ajoutée ou non) soit optionnels (une caractéristique remplacée par une autre).

Table 1 Table de Précédence

Service Id	Condition de Précédence
Service1	-
Service2	Condition1
Service3	Condition2
Service4	Condition3
…	….
Contrainte d'Exclusion Mutuelle	
(Service n) XOR (Service m)	
Choix Modulaires	
(Service i) XOR(Service j)	
Service w	

4.3.3 Approche orientée-produit

Plutôt que de publier une liste d'instances de services identifiée à partir des caractéristiques produit, l'approche orientée-produit identifie des modèles élémentaires de RdP à partir de chacune de ces caractéristiques. A partir de la table de précédence, des techniques de synthèse (Jeng and DiCesare, 1990) peuvent être utilisées pour construire le modèle RdP général du produit. Cette dernière étape peut facilement être réalisée en fusionnant des places avec une technique classique telle que « *1-way merge* » (Dicesare et al., 2012).

Les conditions de mutuelles exclusions sont traitées de manière légèrement différente. Ces contraintes sont appliquées sur les différentes alternatives pouvant produire la même caractéristique. Elles sont ajoutées au premier service de chaque alternative. Normalement, une utilisation stricte des méthodes de modélisation *SOP* ne devrait pas nécessiter l'utilisation d'alternatives, puisque les services de production modélisent la transformation de l'état du produit sans considération sur la méthode de transformation, laissant cette décision aux ressources. Toutefois, il peut s'avérer pratique de construire des modèles *SOP* basés sur des ontologies de services mixtes, i.e. avoir la possibilité de modéliser chaque alternative avec une ontologie différente.

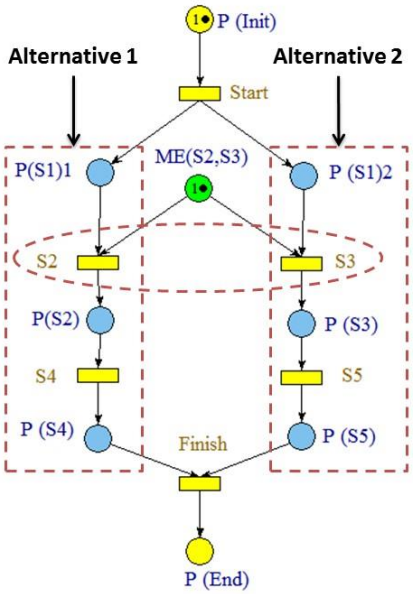


Figure 47 Alternatives dans un modèle *SOP* en RdP

4.4 GENERATION DU MODELE

4.4.1 Relations de précédence

Avec l'adoption de la perspective de prédécesseurs, les relations de précédences rencontrées dans un système de production peuvent être catégorisées en 8 types:

1. *Précédence Simple:* un service *A* est exécutable après qu'un service *B* soit terminé.
2. *Précédence Composée en ET:* un service *A* est exécutable après qu'un service *B* et un service *C* soient terminés.
3. *Précédence Composée en OU:* un service A est exécutable quand soit un service B soit un service C sont terminés.
4. *Précédence Simple Partagée*: une précédence simple qui apparait dans la condition de précédence d'autres services dans la table de précédences.

5. *Précédence Composée en ET Partagée:* une précédence composée en ET qui apparait dans la condition de précédence d'autres services dans la table de précédences.
6. *Précédence Composée en OU Partagée:* une précédence composée en OU qui apparait dans la condition de précédence d'autres services dans la table de précédences ;

Toutes ces conditions de précédence doivent être écrites sous forme canonique minterme, i.e. en tant qu'une somme de produits booléens. Les précédences simples et composées en AND représentent des mintermes tandis que les précédences composées en OU représentent des maxtermes. Ce point permet une meilleure compréhension des deux dernières relations de précédence:

7. *Précédence d'Exclusion:* relation de précédence ayant la forme de n'importe laquelle des relations précédentes mais incluant une contrainte d'exclusion vis-à-vis de ses prédécesseurs. Un service n'est exécutable que si un ou plusieurs services n'ont pas été exécutés auparavant ;
8. *Précédence d'Exclusion Mutuelle:* quand deux services s'excluent mutuellement dans leurs relations de précédence.
9. La section suivante présente la règle de composition de ces relations de précédence.

4.4.2 Règles de modélisation

Après l'analyse exhaustive des relations de précédence, un ensemble de règles de modélisation sont proposées pour permettre la modélisation de ces relations grâce au formalise des RdP de manière compacte mais facile à lire et comprendre. Les règles suivantes permettent de construire de tels réseaux à partir de la table de précédences.

Règle No.1 Les conditions de précédence doivent toujours être exprimées dans sa forme canonique minterme comme une somme de produits.

Exemple: Une condition telle que *S2(S1|S3)* doit être exprimée dans la table en *(S1*S2)|(S2*S3)*.

Règle No.2 Pour chaque service entré dans la table, une transition est créée.

Corollaire 2.1 Pour chaque service ayant une condition de précédence exprimée en somme de mintermes, chaque minterme doit être modélisé séparément comme une nouvelle entrée de la table en conservant le même label de service. Les copies de label de service générées sont appelés services clonés.

Corollaire 2.2 Pour toutes les transitions ayant un label de service cloné, une seule Place Locale d'Exécution est créée et reliée en tant que place d'entrée avec un seul arc pondéré.

Corollaire 2.3 Chaque entrée de service cloné dans la table est modélisée avec les mêmes règles et conditions mintermes, conditions d'exclusion et d'exclusion mutuelle.

La Figure 48 illustre la règle 2 et ses corollaires. Dans la table de précédence, le service *S4* est séparé en deux entrées *S4*, une pour chaque minterme, et une nouvelle table est créée. Après séparation, toutes les entrées de services créent une transition dénommée avec le nom du service. Comme le service *S4* est cloné, une place d'exclusion locale est ajoutée pour ne limiter qu'à une fois son exécution.

Service Id	Condition de Précédence
S1	-
S2	-
S3	S1
S4	(S1*S2) | (S2*S3)

Service Id	Condition de Précédence
S1	-
S2	-
S3	S1
S4	(S2*S3)
S4	(S1*S2)

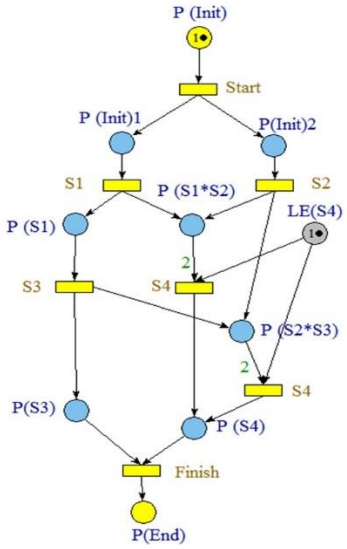


Figure 48 Exemple illustratif n°1

Règle No.3 Pour chaque occurrence d'un service dans une précédence simple, une place de permission est créée pour servir de permission à chaque successeur.

Corollaire 3.1 Les places de permission sont reliées comme entrées pour les instances de service dépendantes.

En reprenant l'exemple de la Figure 48 , le service *S1* apparait comme ayant une précédence simple au service *S3*, donc *P(S1)* est créée. De la même manière, un prédécesseur nul apparait en précédence simple des services *S1* et *S2*, donc *P(Init)1* et *P(Init)2* sont créées.

Règle No.4 Pour chaque occurrence d'un minterme dans la colonne de conditions de précédence, à place de permission est créée et dénommée comme le minterme.

Corollaire 4.1 Les places de permission sont reliées en entrées de chaque instance de services dépendante.

Corollaire 4.2 A chaque place de permission créée, les arcs de sortie (PxT) reliant aux services successeurs ont un poids égal au nombre d'éléments inclus dans le minterme.

Cette règle s'applique aux conditions de précédence de chaque occurrence du service cloné *S4*. Les places de permission *P(S1*S2)* et *P(S2*S3)* sont créées pour les conditions minterme et sont reliées en tant que places d'entrées aux transitions correspondantes, toute les deux dénommées *S4*. Comme chacun de ces mintermes contient 2 éléments, l'arc connectant les places de permission à leur transition a un poids de 2 de manière à extraire tous les jetons et éviter une exécution répétée.

Règle No. 5 Pour toutes les conditions de précédence exprimées sous forme de maxterme et présentant *n* occurrences dans la colonne des conditions de précédence, seule une place de permission est créée.

Corollaire 5.1 La place de permission est reliée en entrée de tous les services successeurs avec des arcs de lecture de poids unitaire.

Corollaire 5.2 Pour tous les services ayant un maxterme en tant que place d'entrée, une place d'exclusion locale est créée et reliée en entrée.

Comme on peut le voir dans l'exemple de la Figure 49 , les services *S3* et *S5* ont des conditions maxterme identiques. De ce fait, seule une place de permission *P(S1|S2)* est créée et reliée à la fois à *S3* et *S5* avec des arcs de lecture. De plus, pour limiter la réexécution du service une fois exécuté, les places d'exclusion locale *LE(S3)* et *LE(S5)* sont créées.

Service Id	Condition de Précédence
S1	-
S2	-
S3	(S1 | S2)
S4	(S1*S3) | (S1*$\overline{S2}$)
S5	(S1 | S2)

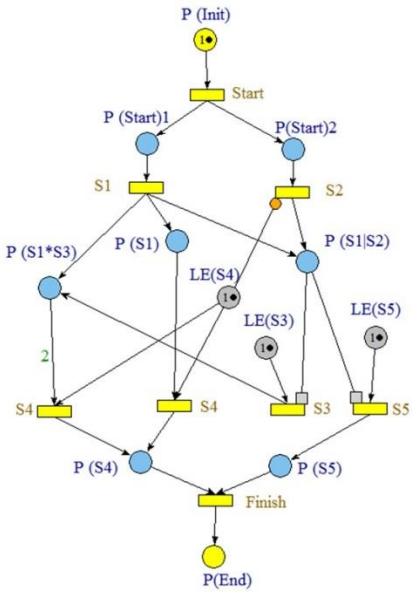


Figure 49 Exemple illustratif n°2

Règle No.6 Chaque condition minterme contenant des éléments de négation doit être découpé en mintermes complémenté et un minterme non-complémenté.

Règle No.7 Les conditions minterme complémentées ne créent pas de nouvelles places de permission. A la place, une place d'exclusion locale est créée pour les services successeurs si elle n'est pas déjà créée.

Corollaire 7.1 La place d'exclusion locale du service dépendant est relié avec des arcs inhibiteurs à toutes les transitions dénommées par les services apparaissant dans le minterme complémenté, sauf pour les services à mutuelle exclusion (voir la règle 8).

Les règles 6 et 7 apparaissent dans la Figure 49 au sujet du service *S4*. Le minterme *(S1*$\overline{S2}$)* est découpé, modélisant *S1* comme une précédence simple. Comme S4 a déjà une place d'exclusion locale associée, la partie complémentée, *($\overline{S2}$)*, relie *LE(S4)* à *S2* avec un arc inhibiteur. Cela limite l'exécution de *S2*, et autorise seulement une fois *S4* à être exécuté.

Règle No.8 Pour deux services étant en condition d'exclusion mutuelle, une place d'exclusion mutuelle est créée et est reliée par des arcs à poids unitaires en tant qu'entrée des transitions associées aux mintermes contenant ces conditions d'exclusion mutuelle.

La Figure 50 illustre une situation d'exclusion mutuelle entre les services *S3* et l'un des clones de *S4*. Il y a donc une place d'exclusion mutuelle créée et reliée à *S3* et à l'instance clonée de *S4* excluant *S3*. La situation d'exclusion mutuelle est donc simplement considérée avec ce clone de *S4*. Toutefois, *S3* est aussi modélisé avec une relation d'exclusion avec l'autre clone de *S3* (conformément aux règles 6 et 7).

Service Id	Condition de Précédence
S1	-
S2	-
S3	$S1*\overline{S4}$
S4	$(S2*\overline{S3})$
S4	(S1*S2)

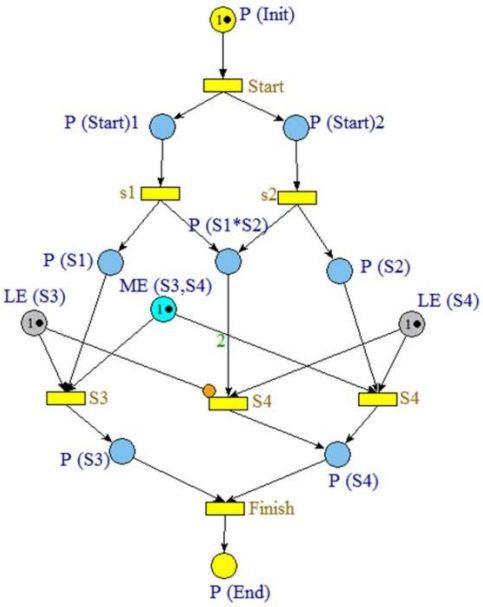


Figure 50 Exemple illustratif n°3

Règle No.9 Tous les services n'apparaissant pas dans la colonne de précédence créent une seule place de permission.

Corollaire 9.1 Une transition factice « *Finish* » est créée, indiquant, quand activée, la conclusion du processus de production du produit. Toutes les places de permission créées par la règle 9 sont reliées à cette transition.

Comme remarqué dans l'exemple précédent, il n'y a pas de service dépendant de l'exécution ni de *S3* ni de *S4*, donc des places de permission sont créées pour chacune et sont reliées à une transition factice dénommée « *Finish* » qui sera activée à la fin de la production.

4.4.3 Algorithme de génération

L'ensemble des règles présentées ci-dessus peuvent être utilisées dans un algorithme de génération automatique du RdP à partir du tableau de précédence. Ce RdP, avec les règles d'évolution de son formalisme, capture toute la structure du processus à travers les dépendances entre services. À un état donné sur le cycle de vie du produit, le réseau habilitera l'exécution des services qui satisfait les contraintes de production et appartient à l'espace de solutions séquentielles du produit.

L'ensemble des règles édictées ci-dessus permet de construire l'algorithme pratiquement directement, mais celui-ci n'a pas encore été réalisé. Cet élément fait donc partie des suites directes à ce travail.

4.5 CAS D'ETUDES

Cette section va développer deux cas d'études permettant d'illustrer sur des exemples complets la génération du RdP global de familles de produits déterminés. Le premier cas d'étude s'intéresse à la construction de la table de précédence et à l'ontologie de services liée à l'application, tandis que le second s'intéresse à la génération du RdP en lui-même.

4.5.1 Cas d'étude n°1

Le premier cas d'étude s'intéresse à une compagnie métallurgique imaginaire fabriquant des plaques telles que celle présentée Figure 51 . Tous les produits de cette série forment une famille avec tous ses membres partageant des caractéristiques communes, i.e. une plaque rectangulaire en acier avec deux trous. Toutefois, les plaques se différentient par d'autres caractéristiques telles que: la position et la taille des trous, les dimensions de la plaque (WxLxH), et si les trous ont (i) un pion de centrage assemblé; (ii) un profil taraudé; (iii) un profil lisse. Une flexibilité complète est laissée au client, qui peut personnaliser tous ces paramètres. De tels choix modulaires différentient des sous-familles de produits (versions de produits), en dans ce cas $3^2=9$ sous-familles.

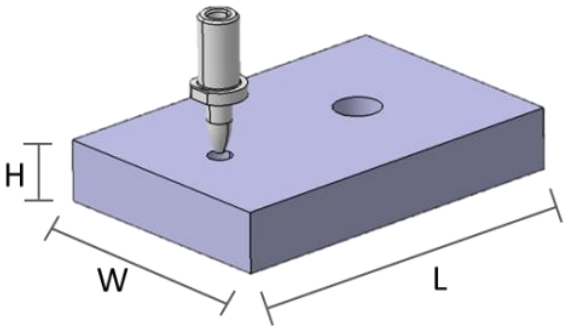


Figure 51 Plaque à 2 trous

Les concepteurs du processus font ensuite une identification des similitudes trouvées dans le domaine physique dans le domaine du processus, comme illustré dans la Figure 52 Une telle identification crée une librairie de services basée sur une ontologie de services propre à l'entreprise.

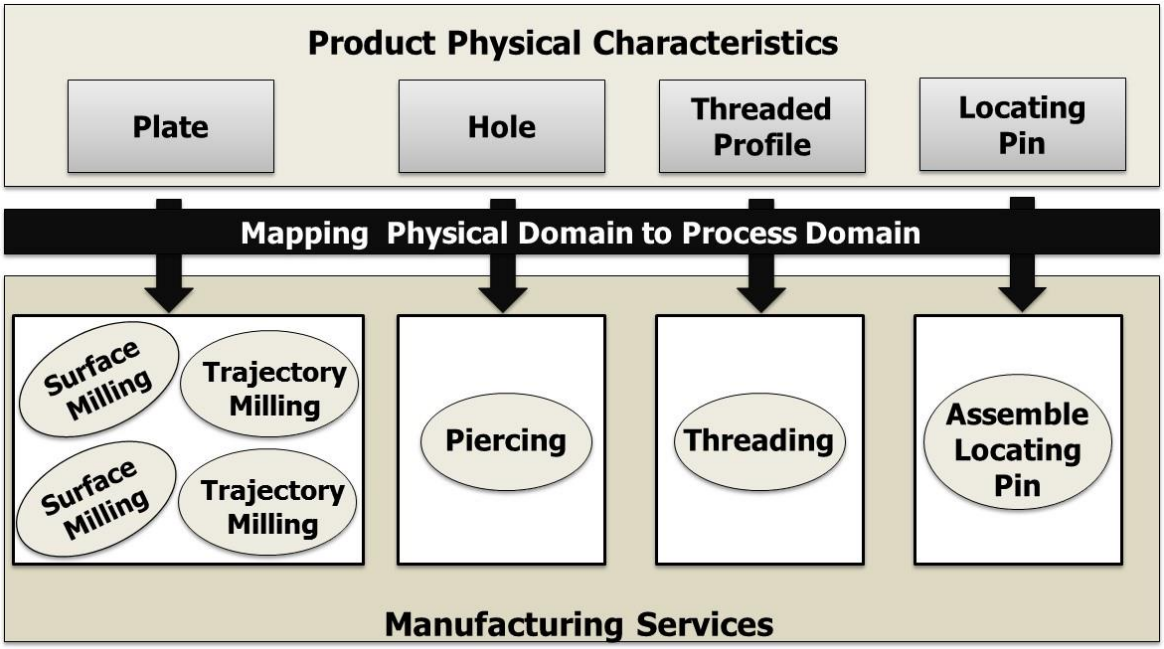


Figure 52 Identification des caractéristiques du domaine physique au domaine du processus

Au sein de l'ontologie de services, la production de cette famille de produits est modélisée par un type de service dénommé « *Produce 2 Hole Plate* ». Un client définit ensuite la spécification de services en définissant les valeurs de tous les paramètres indiqués par le type de services comme illustré dans la Table 2.

Table 2 Specification du Service « *Produce 2 Hole Plate* »

Service Parameters	
Parameter	**Value**
Material	Stainless steel type 102
(H,W,L) dimensions	(0,4cm,4cm,8cm)
Hole1_(x,y) coordinates	(2cm,2cm)
Hole 1_Diameter	4,826mm (UTS type10)
Hole 1_Locating Pin	none
Hole 1_Threaded Profile	UTS type 10
Hole 2_(x,y) coordinates	(2cm,6cm)
Hole 2_Diameter Hole2	6mm
Hole 2_Locating Pin	6,3mm
Hole 2_Threaded Profile	none

Pour simplifier, la table de précédence de cet exemple a été définie en représentant une séquence unique en définissant uniquement un prédécesseur par service, Table 3 . Néanmoins, les vraies relations de services proposeraient plusieurs séquences de production. Des contraintes modulaires sont également considérées dans la table de précédence. Par exemple, le service « *Thr.1* » est seulement possible si le service "*Pr.2*" a été réalisé et que le choix modulaire *Ch.1* a été défini à « *Thr.1* », indiquant l'ajout d'un taraudage sur le trou 1.

Table 3 Table de Précédence du Service : « *Produce 2 Hole Plate* »

MService Type	Instance ID	Precedence Condition
Surface Milling	Surf_Mill.1	-
Surface Milling	Surf_Mill.2	Surf_Mill.1
Trajectory Milling	Traj_Mill.1	Surf_Mill.2
Trajectory Milling	Traj_Mill.2	Traj_Mill.1
Piercing	Pr.1 *(hole 1)*	Traj_Mill.2
Piercing	Pr.2 *(hole 2)*	Pr.1
Threading	Thr.1 *(hole 1)*	(Ch.1=*Thr.1)*& Pr.2
Threading	Thr.2 *(hole 2)*	(Ch.2=*Thr.2)*& (Thr.1|(Pr.2*Ch.1=$\overline{Thr.1}$))
Assembly Locating Pin	ALP.1 *(hole1)*	(Ch.1=*ALP.1)*& (Thr.2 | (Pr.2*Ch.2=$\overline{Thr.2}$))
Assembly Locating Pin	ALP.2 *(hole2)*	(Ch.2=*ALP.2)*& (ALP.1|(Thr.1* Ch.1 = *Thr.*1)|(Pr.2* Ch.1= *null.1*))
Modular Constraints		
Ch.1= *Thr.1|ALP.1|null.1*		
Ch.2= *Thr.2|APL.2|null.2*		

Le modèle de processus du service « *Plaque 2 Trous* » est ensuite construit et affiché dans la Figure 53 . Il contient une liste des services de production niveau Produit, une liste des spécifications des paramètres de processus et une liste de dépendances de services. Chaque service possède une liste de spécifications de paramètres identifiée par la classe *MethDeProces* parmi les spécifications de paramètres du niveau supérieur. Par exemple, le paramètre « "*Milling Depth* » du service « *Surface Milling* » est défini par la dimension en hauteur de la plaque et le paramètre « *Turning Speed* » est défini en fonction du matériau du produit.

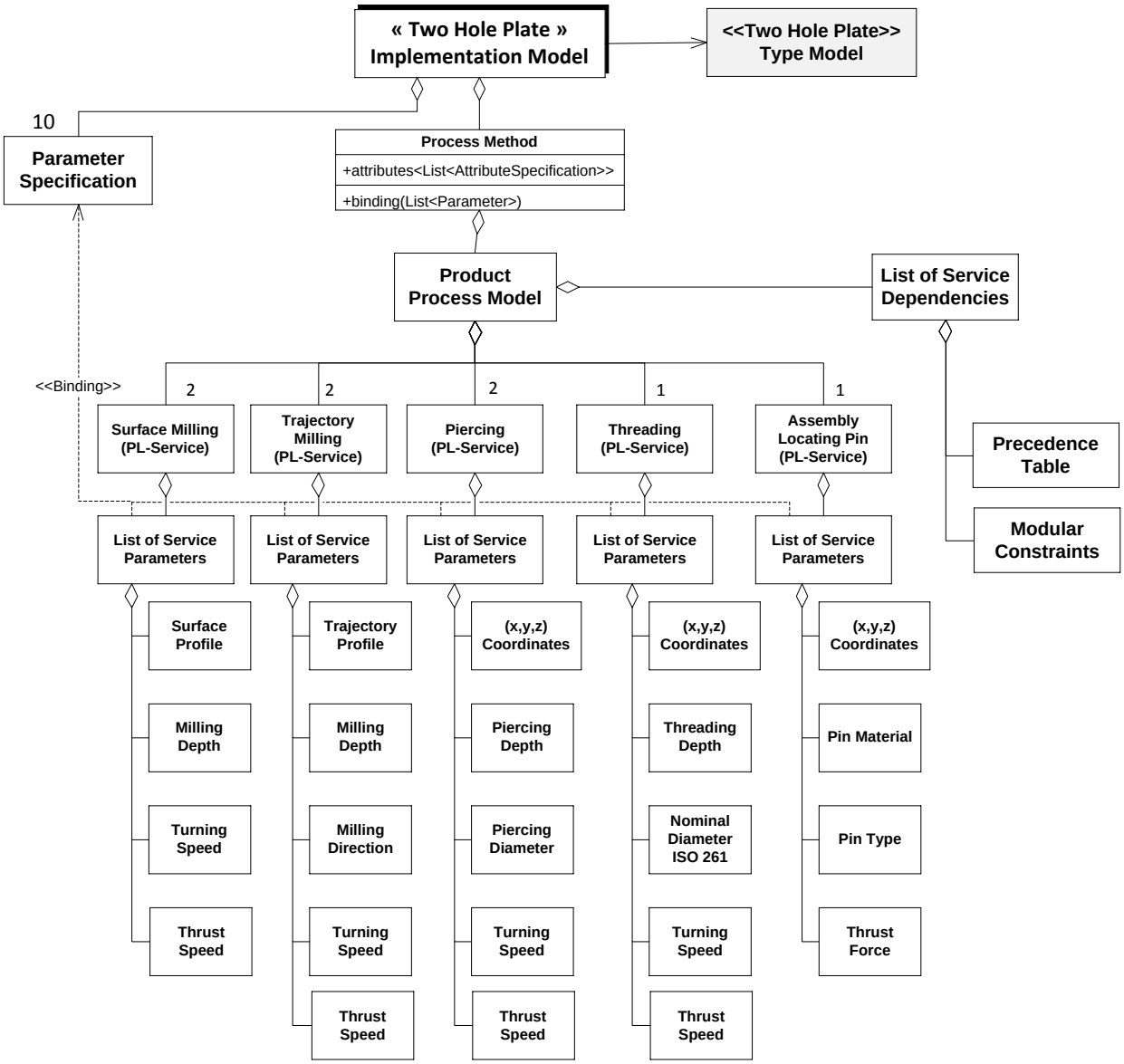


Figure 53 Modèle *ImpMServ* du service : « *Plaque 2 Trous* »

4.5.2 Cas d'étude n°2

Ce second cas d'étude est construit autour de blocs Lego®. Ces blocs se trouvent être une bonne illustration des dépendances exprimées précédemment entre caractéristiques du produit, qui est donc un assemblage de blocs. La structure Lego® est de plus utilisée comme une référence visuelle des précédence entre blocs.

Le produit étudié est une structure de blocs Lego® formée d'un maximum de 4 couches et trois types de blocs, i.e. un petit bloc 2x2, un bloc moyen 2x4 et un long bloc 2x6. Chaque bloc est disponible en 4 coloris {rouge, vert, jaune, bleu} et peut être assemblé dans n'importe quelle position sur une base ou sur d'autres blocs tant que l'interface inférieure permet une stabilité de l'assemblage. Pour l'illustration, les blocs Lego® sont directement utilisés pour représenter les instances de services requises pour former le processus de production du produit. Le type de service est caractérisé dans cet exemple par le couple {type de bloc ; couche}, ce qui donne une ontologie de 4x3 = 12 types de services : *A1, B1, C1, A2, B2, C2, A3, B3, C3, A4, B4, C4* (Figure 54). Les paramètres de services sont donc la couleur, la position dans les coordonnées x et y et l'orientation (axe w) pour les blocs *B* et *C*.

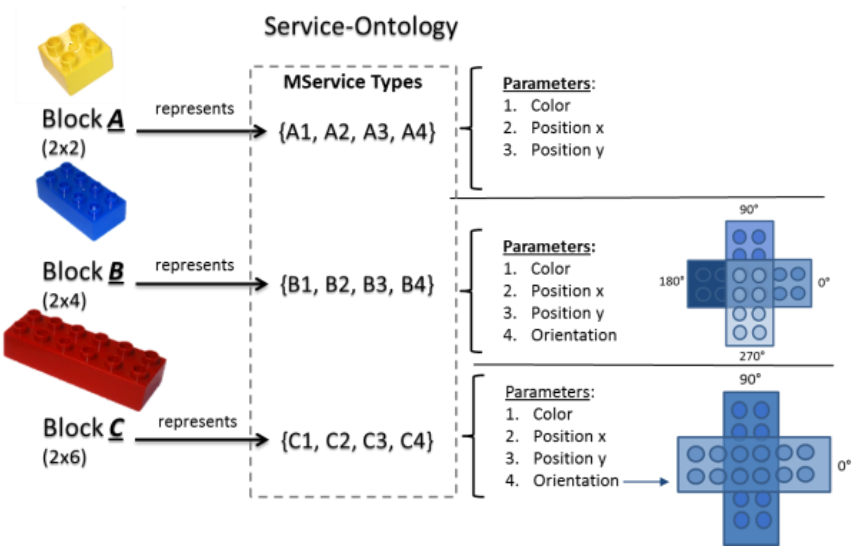


Figure 54 Ontologie de Services du Cas d'Etude n°2

La Figure 55 présente la famille de produit de ce cas d'étude. Le choix modulaire *(A4.3*A5)* ***XOR*** *(A4.1*A4.2)* indique que le concepteur du produit, qui peut être par exemple un client, doit faire un choix pour soit inclure les caractéristiques induites par les services *A4.3* et *A5* ou celles induites par les services *A4.1* et *A4.2*. Chaque choix représente une variante dans la famille de produit.

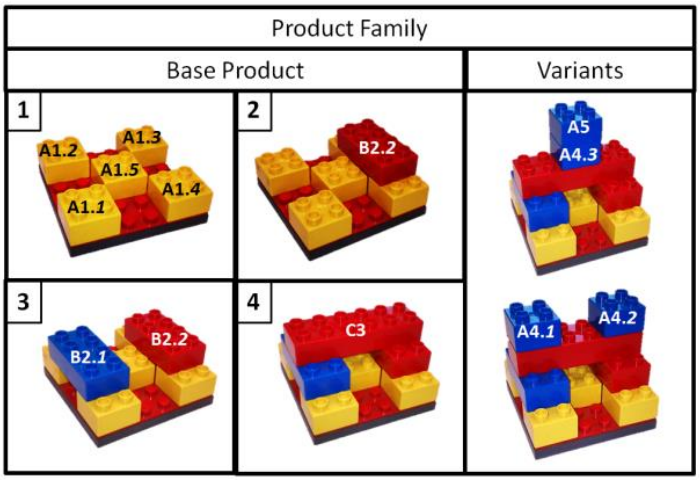


Figure 55 Famille de produit étudié

Pour créer le modèle *SOP* de ce produit, la première étape est donc de définir sa table de précédence. Figure 56 illustre la démarche complète et les résultats obtenus. Comme décrit précédemment, une table préliminaire peut être générée directement à partir de considérations géométriques du produit. On y retrouve toutes les considérations sur quels blocs peuvent être posés avant quels autres, avec des contraintes basées sur l'accessibilité des blocs par le haut, l'outil de dépôt étant d'axe vertical. Une seconde étape consiste à enrichir cette table avec d'autres contraintes opératives nécessitant cette fois l'intervention de l'opérateur. Dans cet exemple, une contrainte d'accessibilité a été ajoutée pour les services *B2.2* et *B2.1*. Il est supposé que, à cause de la forme et de la taille de la pince de dépôt du bloc *A1.5*, elle ne pourra plus accéder à la position de dépôt une fois que les services *B2.2* et *B2.1* auront été tous les deux placés, comme illustré sur la Figure 57 . Toutefois, si seulement l'un des deux services a été exécuté auparavant, alors la position est accessible par une rotation de la pince. Cette contrainte est modélisée avec une contrainte de précédence d'exclusion mutuelle entre les deux services et *A1.5*. Cette modélisation n'est pas unique, et l'on pourrait par exemple imaginer de contraindre *A1.5* plutôt que *B2.2* et *B2.1*, ce qui serait totalement valide par rapport aux règles de modélisation.

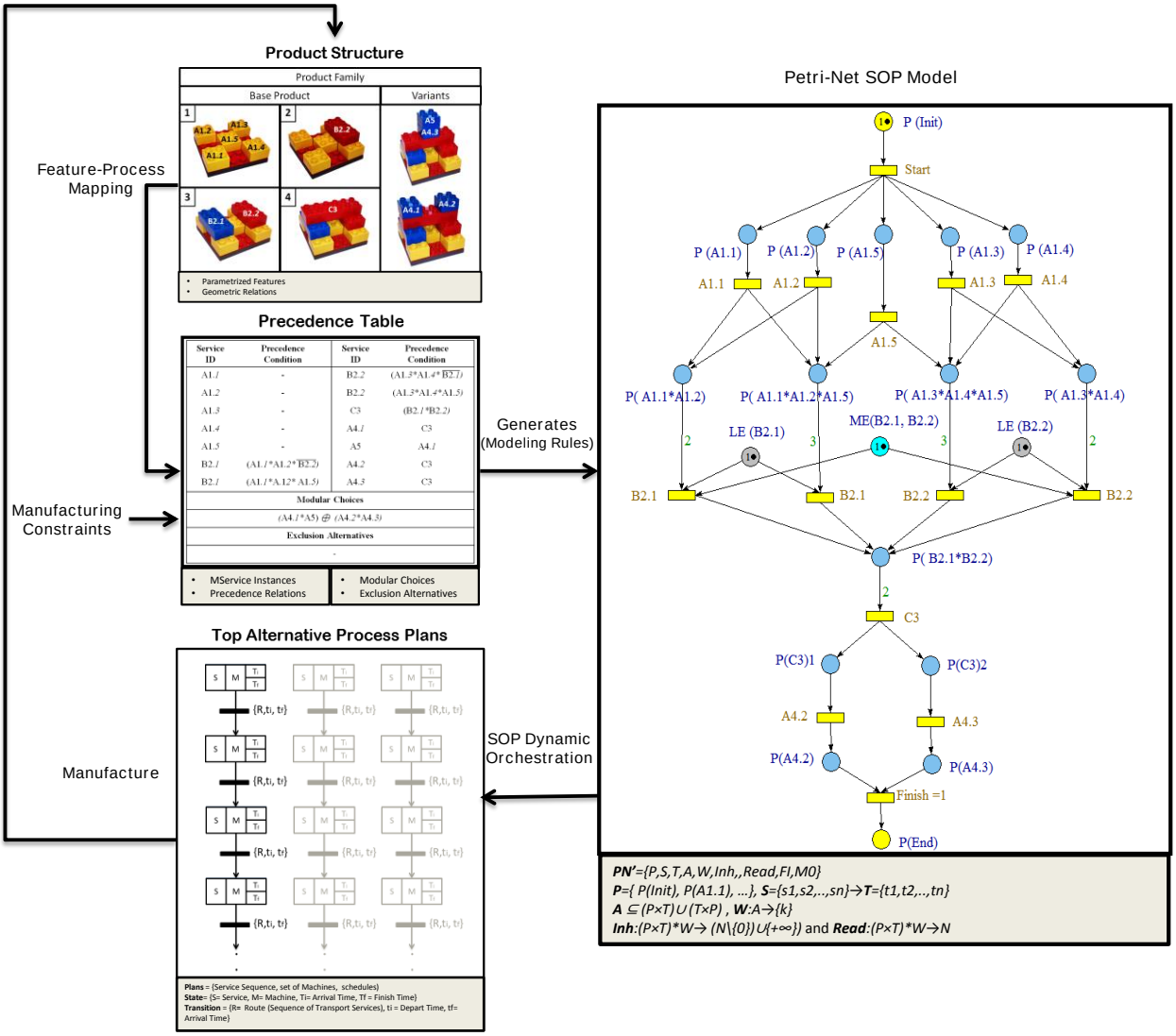


Figure 56 Modèle de processus orienté-services en RdP

La Figure 57 illustre l'évolution du réseau à cause de cette contrainte d'accessibilité. A l'état *1*, les services *A1.1* à *A1.4* ont été correctement exécuté, mais pas le service *A1.5*. Cette situation permet un clone du service *B2.1* et un clone du service *B2.2* étant donné que *ME(B2.1*B2.2)* contient encore un jeton. Puis, en imaginant l'exécution de *B2.2*, les jetons de *ME(B2.1*B2.2)* et *LE(B2.2)* seront consommés, et désactiveront donc *B2.1* et l'autre clone de *B2.2* (état 2). La seule manière d'activer une transition et de permettre l'évolution est alors d'exécuter *A1.5*, ce qui fournira le jeton manquant au second clone de *B2.1* (état 3). Finalement, *B2.1* est exécuté et tous les clones de *2B.1* et *2B.2* sont désactivés. Ainsi, les séquences possibles (en tenant compte de ces trois services) sont {1.5→2.1→2.2}, {1.5→2.2→2.1}, {2.1→1.5→2.2} et {2.2→1.5→2.1}, ce qui satisfait la contrainte d'accessibilité de l'outil.

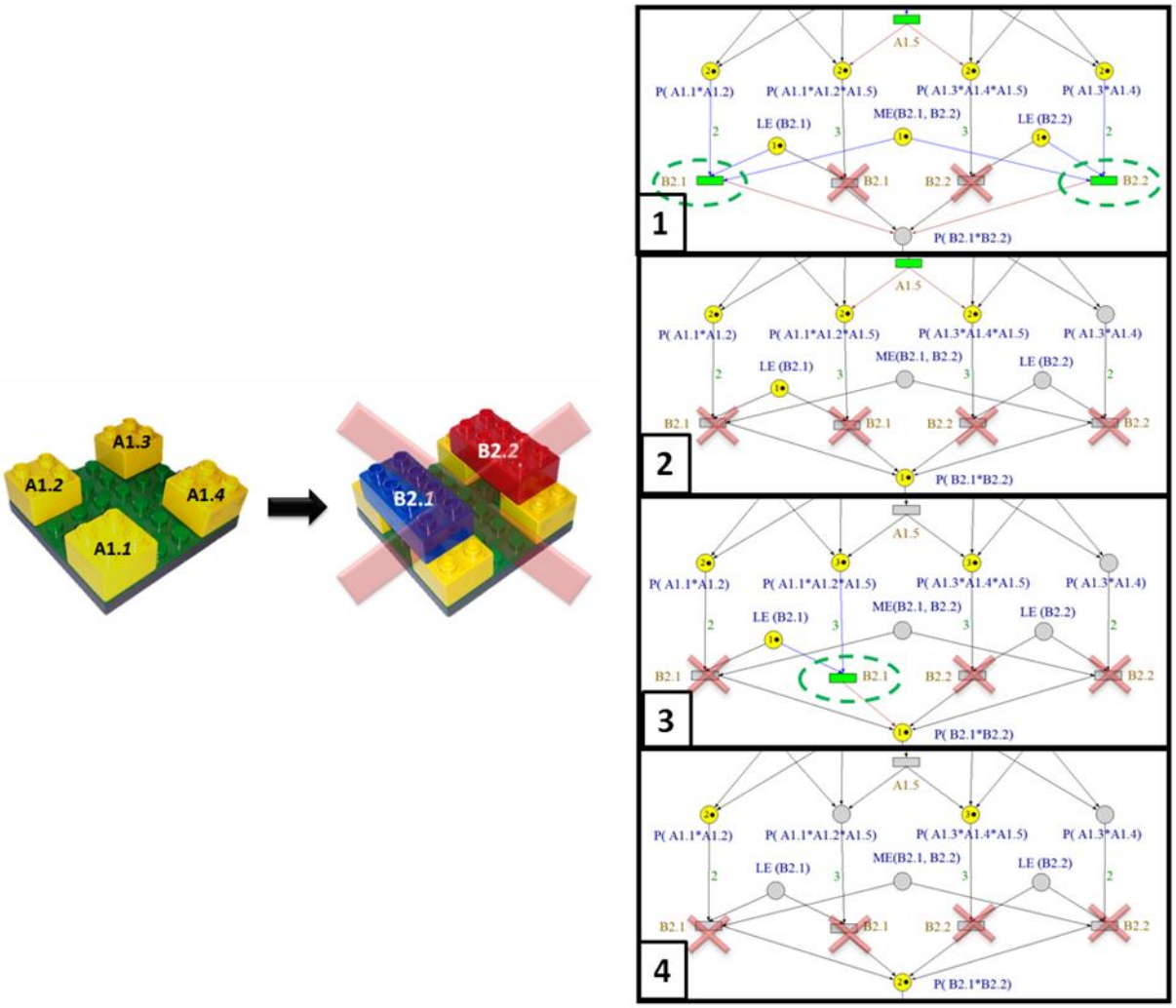


Figure 57 Séquence de dépôt interdite

La troisième étape consiste à indiquer les choix modulaires, puis à générer le RdP avec tous les services inclus pour la variante de produit. La Figure 56 présente le modèle en RdP de la variante 2 de la famille de produits. Ce réseau a été construit en utilisant l'outil de modélisation Roméo (Gardey, G et al., 2005; Lime, D et al., 2009), développé au laboratoire IRCCyN. Avec ses arcs inhibiteurs et de lecture, le réseau peut être simulé, et un graphe d'état peut être généré. Avec un algorithme de recherche en profondeur, le nombre de séquences possibles représentées par ce réseau peut être calculé. Pour la version 1, l'espace de solutions séquentielles se monte à un nombre de 480 séquences faisables.

4.6 CONCLUSION

Ce quatrième chapitre avait pour objectif de présenter un formalisme permettant de tirer parti au maximum de l'introduction du concept de services dans la construction des gammes de production. Afin de les rendre les plus flexibles possibles, tout en conservant une représentation complète et facilement manipulable par à la fois les pilotes de systèmes de production et les systèmes informatiques d'aide au pilotage, nous avons proposé l'utilisation du formalisme RdP.

Après avoir présenté les différentes alternatives présentes dans la littérature provenant de la communauté productique, nous avons mis en avant les propriétés attendues d'une bonne utilisation des RdP dans la modélisation des gammes. Pour cela, nous avons défini une méthodologie complète de modélisation, permettant au concepteur de modéliser sa gamme en termes de précédences uniquement. Un ensemble de primitives en RdP a également été défini, permettant d'associer à chaque situation pouvant être

rencontré dans la définition des gammes une primitive prédéfinie. Nos travaux ont permis de proposer tous les éléments nécessaires à la conception d'un algorithme de génération automatique permettant de passer directement du tableau de précédence à un RdP utilisable. Enfin deux cas d'étude sont présentés, permettant de mettre en avant les propriétés attendues de l'utilisation de gammes flexibles, notamment en termes d'ouverture de l'espace de solutions utilisables par le système de contrôle.

Après la définition de ces modèles de gamme, l'utilisation qui en est faite est directement liée au concept d'orchestration des plans de production. Le chapitre suivant présente ce que nous proposons d'utiliser en tant que processus liés à l'orchestration des plans de production au sein du *SoHMS*.

Chapitre 5

Orchestration des Plans de Production Orientés-Services

À ce niveau du document, un système holonique pour lequel on a intégré les concepts et principes de conception SoA à travers un modèle conceptuel et computationnel de services de production a été défini. Ensuite, un modèle computationnel a été proposé pour la spécification des gammes de production de produits flexibles avec le but de permettre l'exploration d'un espace de solutions plus large pour ainsi incrémenter la flexibilité et la réactivité du système de contrôle face à des situations dynamiques.

Cependant, un modèle de gammes flexibles n'est qu'un porteur de la flexibilité, il convient aux outils de planification et d'orchestration d'exploiter cette flexibilité. Dans ce chapitre, on présente un processus pour l'orchestration des plans de production à partir des modèles de gammes flexibles présentés dans le chapitre précédent. Dans la première partie de ce chapitre, on commence avec une définition des activités de contrôle, soit la planification et l'ordonnancement, pour bien localiser les apports vis-à-vis de ces activités, ainsi qu'une introduction à l'approche de la littérature «Integrated processus planning and Scheduling» (IPPS) proposant une fusion de ces deux activités pour générer des plans de production plus adaptés à l'état du système. Dans la deuxième partie, le processus d'orchestration dans un SoHMS *est présenté, décrivant les interactions holoniques nécessaires pour la définition des plans de production. Pour répondre à la possible émergence d'une grande complexité combinatoire, dans la troisième partie, des stratégies pour l'orchestration sont proposées. Puis, dans la dernière partie, un exemple illustratif du protocole d'orchestration avec l'application des stratégies est présenté.*

5.1 INTRODUCTION : PLANIFICATION ET ORDONNANCEMENT DES PROCESSUS

5.1.1 Planification de processus

La *Planification de Processus* (PdP) de fabrication est l'activité de production qui consiste à déterminer la manière avec laquelle un produit sera fabriqué. En détail, c'est le processus de sélection et de séquencement des opérations et la définition des paramètres nécessaires à la réalisation d'un produit tout en achevant des objectifs spécifiques, notamment de réduction des coûts et de temps de production. Une autre définition apportée par (Chang and Wysk, 1985) : *« La Planification des Processus est l'acte de préparer des instructions détaillées d'opérations pour transformer une conception d'ingénierie vers un produit final »*. L'étape de PdP s'occupe alors de la transformation des informations relatives à la conception de produit en étapes de production pour une production efficiente et efficace. L'issue de cette étape est, typiquement, une fiche

des gammes portant des informations sur le routage lors de la fabrication, les opérations, les paramètres de ces opérations, les machines et les outils dont a besoin la production.

Cependant, créer des plans de processus pour des applications avec une grande variété de produits n'est pas une tâche facile à réaliser ni à consolider. Comme (Crow, 1992) le remarque, « *La PdP manuelle se base sur les expériences des ingénieurs de production et leurs connaissances sur les équipements, leurs capacités, méthodes et outils. La PdP nécessite un grand investissement en temps, et les résultats sont très dépendants de la personne effectuant la planification.* » Ce sont ces complications qui ont motivé la création des systèmes CAPP (Systèmes de Planification de Processus Assistés par Ordinateur). Leur but est de créer automatiquement des plans de production à partir des modèles CAO des produits. Parmi les avantages amenés par l'utilisation de systèmes CAPP, on trouve : (i) une réduction des compétences du planificateur, (ii) une réduction du temps du processus de planification, (iii) une réduction des coûts de production, (iv) des plans plus cohérents et précis et (v) une productivité augmentée.

Il existe deux approches pour les systèmes CAPP :

- *Approche par Variantes* : L'approche se base sur la technologie des groupes et les requêtes de bases des données. Cette approche consiste à standardiser les processus de l'ensemble des produits et familles de produits pour ainsi créer une base de données, à partir de laquelle on cherche à extraire les processus permettant de former un plan de production. Ainsi, quand un nouveau composant de produit s'intègre à la production, un processus similaire créé auparavant est récupéré depuis la base des données et est modifié pour s'adapter au nouveau composant. La plupart des systèmes CAPP peuvent être classées dans cette catégorie (Shen et al., 2006).
- *Approche Générative* : Cette approche est centrée sur la génération automatique des processus de production optimaux à partir de la composition physique (plan des composants). Les systèmes de ce genre utilisent des bases des données de connaissances, des descriptions des composants et d'une logique de raisonnement pour synthétiser des plans de production cohérents. Cependant, dû à la complexité pour représenter toute l'information des composants et leurs interactions, un véritable système génératif capable de satisfaire les besoins industriels n'a pas encore été réellement développé (Shen et al., 2006).

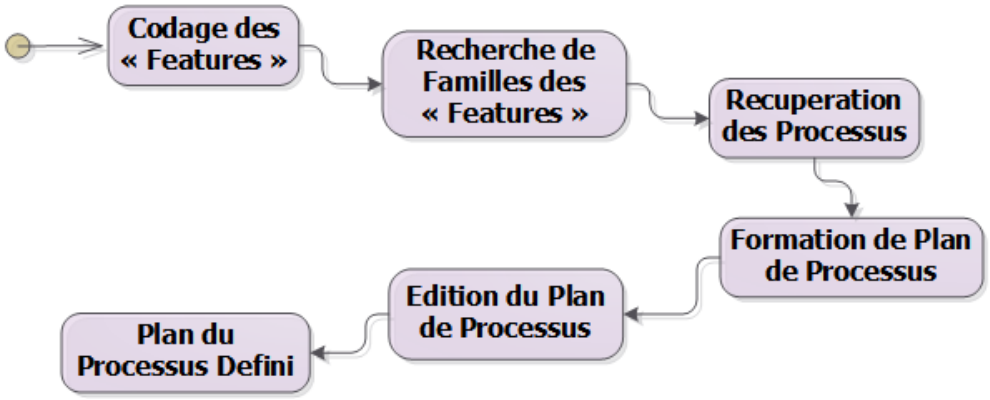


Figure 58 Planification de Processus : Approche par Variantes

5.1.2 Ordonnancement

L'ordonnancement au niveau de l'atelier est le processus d'allocation des ressources d'atelier et des opérations sur des intervalles de temps. Cette activité correspond principalement à déterminer la date la plus appropriée pour l'exécution d'une tâche, prenant en compte les contraintes temporelles parmi les opérations (précédence) ainsi que la disponibilité des ressources et des matériaux. Le problème d'ordonnancement peut être classé en différents types selon quatre paramètres : les modèles d'arrivée des

ordres, le nombre des machines-ressources dans l'atelier, le type de routes dans l'atelier et le critère avec lequel chaque ordonnancement sera évalué (Conway et al., 1967).

Les assignations faites durant cette étape ont une grande influence sur la performance du système en termes de coût, délais et cadence. De ce fait, le processus d'orchestration peut être considéré comme une étape d'optimisation où un ensemble limité de ressources et attribué au travers du temps à des activités parallèles comme séquentielles (Zweben and Fox, 1994). De plus, on peut ajouter de la flexibilité au système au travers de stratégies de ré-ordonnancement, lesquelles, selon le degré de flexibilité apporté par l'architecture de contrôle, peuvent augmenter la réactivité du système.

Un problème typique d'ordonnancement intéressant ce document est un problème type Job-shop. Celui-ci se formule comme un ensemble de processus, appelés jobs, et un ensemble des ressources-machines. Chaque ressource-machine peut au plus exécuter un job à la fois et chaque job consiste en une suite d'opérations non-interruptibles exécutables dans une ressource-machine donnée. Ce type de problème est connu pour être NP-difficile. De ce fait, les investigations menées dans le domaine ont été ciblées pour la création d'algorithmes efficients pour attaquer ce type de problèmes, qui déjà pour des tailles modérées d'environ 10 jobs et 10 machines devient un challenge pour la communauté (Tan and Khoshnevis, 2000). Le problème devient encore plus complexe à attaquer dans les situations suivantes :

- Quand en plus des machines, il faut considérer la disponibilité d'autres types de ressources comme des opérateurs ou bien d'outillages partagés ;
- Quand le processus de planification et d'ordonnancement se font en même temps, la qualité des plans peut mener à des solutions optimales mais augmente aussi l'espace des solutions de manière importante ;
- Quand le caractère dynamique du système est important. Dans un système de production, notamment de type Job-shop, rarement les plans peuvent s'exécuter comme prévu. Plusieurs types d'aléas peuvent subir lors de l'exécution demandant le ré-ordonnancement des plans constamment. Parmi ces aléas, on peut trouver : (i) l'indisponibilité des ressources, (ii) l'apparition des nouvelles ressources, (iii) les variations dans le temps de traitement et de début d'exécution des opérations, (iv) les pannes d'alimentation, (v) les défaillances machines, (vi) l'absence des opérateurs, (vii) l'indisponibilité des outils et ou matériaux, etc.

Plusieurs méthodes ont été étudiées pour donner une solution au problème d'ordonnancement. Parmi les méthodes plus typiques, on trouve les méthodes heuristiques, les méthodes de propagation de contraintes, les formalismes de satisfaction de contraintes, la recherche Tabou, les algorithmes génétiques, la logique floue, les réseaux de neurones, etc. (Zweben and Fox, 1994). Les méthodes heuristiques sont des méthodes de recherche locale utilisant des hypothèses basées sur une certaine expérience pour ainsi accélérer le processus de convergence vers une solution satisfaisante. Ces méthodes sont spécialement utilisées pour trouver des solutions non optimales mais acceptables pour les objectifs immédiats. Les méthodes de recherche Tabou est une méthode de recherche locale itérative explorant l'espace voisin d'une solution en recherchant des solutions plus performantes. Pour éviter de retomber dans des minimums locaux, la méthode utilise alors de la mémoire pour éviter de revenir sur des points déjà visités. Les algorithmes génétiques sont des méthodes de type évolutionniste inspiré par le mécanisme d'évolution des organismes du monde naturel. A travers des techniques de mutation, sélection et croisements des paramètres des solutions, l'algorithme parvient à trouver des solutions en apportant des petites améliorations aux solutions plutôt que de faire des grands sauts dans l'espace de solutions. Les algorithmes de type génétique sont spécialement utilisés pour trouver de bonnes solutions globales aux problèmes lorsqu'il n'existe pas de solution exacte.

Cependant, toutes les méthodes traditionnelles, soit heuristiques ou méta-heuristiques comme les méthodes tabou et génétique, ont des difficultés à trouver des solutions aux problèmes réels. Ceci est dû au fait qu'ils ont des méthodes centralisées s'exécutant généralement su un seul processeur. De plus, leurs solutions manquent de flexibilité de par leur nature fondamentale. Les calculs sont faits en se basant sur des hypothèses à propos de l'état du système et négligent les aléas, et donc les solutions deviennent sensibles aux perturbations.

D'autres méthodes de ordonnancement tel que les règles locales d'ordonnancement (Pierreval and Mebarki, 1997) offrent un approche plus réactive. Le problème d'ordonnancement est modélisé comme un ensemble de décisions prises dynamiquement pour créer des solutions valides. Des approches comme celle-ci parviennent à prendre en compte nativement l'état réel du système mais ne garantissent pas une solution de qualité. Pour répondre à ce problème, dans le contrôle classique, des initiatives ont proposé des solutions combinant les principes des deux : ordonnancement prédictif et réactif. L'ordonnancement de groupes vise par exemple à séquencer de manière flexible les opérations pendant l'exécution des plans, et garantir en même temps un minimum de qualité dans leurs solutions par rapport au scénario le plus pessimiste. Une étude comparative des performances de cette méthode par rapport aux incertitudes dues aux temps de transport dans un système de production industriel est présenté dans (Cardin et al., 2013).

En vue de l'incapacité des approches centralisées présentées précédemment à attaquer des problèmes complexes de taille importante, les technologies multi-agents proposent une approche novatrice et légère pour le problème d'ordonnancement. Correspondant à une approche essentiellement distribuée, les systèmes multi-agents sont plus flexibles, efficients et adaptables aux problèmes de l'environnement dynamique des systèmes manufacturiers. Parmi leurs avantages apportées à l'ordonnancement, on peut en lister (Shen et al., 2006):

- La computation parallèle des opérations à travers des processeurs multiples rendent les systèmes d'ordonnancement plus efficients et robustes ;
- Leur architecture facilite l'intégration des activités de planification et d'orchestration ;
- Ils permettent la programmation de comportements de ressources capables de donner priorité aux solutions globales issue de la coopération sur les solutions locales.
- On peut générer des ordonnancements en base aux stratégies de négociation en se servant des mécanismes d'information et coordination implémentés dans le système distribué. Ceci permet de mettre en relation les capabilités des ressources manufacturiers et ainsi rendre possible la planification pas seulement au niveau de la chaine logistique mais aussi au niveau atelier comme au niveau Enterprise.
- Les approches de recherche par heuristique ou méta-heuristique, comme ceux présentés ci-dessus, peuvent être adaptés à certains niveaux des processus de prises de décision.

Dans les architectures holoniques, une somme considérable de travaux a été menée en vue de définir des stratégies d'ordonnancement capables de bénéficier d'un environnement plus flexible. Comme exemple, on trouve dans (Kanchanasevee et al., 1997) une adaptation du protocole Contract-Net capable de supporter des contraintes temporelles et traiter avec les conflits d'ordonnancement en allouant de manière dynamique des opérations aux ressources. Un protocole Contract-Net est aussi proposé dans (Sousa and Ramos, 1999) avec un paradigme de résolutions de conflits basé sur des mécanismes de négociation et l'établissement de contrats. Inspirées des organisations biologiques telles que les colonies de fourmis, des stratégies d'orchestration sont proposées dans (Holvoet and Valckenaers, 2006; Verstraete et al., 2008). Ces stratégies consistent en l'exploration de routes alternatives de production avec une prévision de l'état du système à court terme. Une telle prévision est faite en utilisant le principe de *Stigmergie* come moyen de communication indirect, collectant les informations d'utilisation future du

système. Dans le même sujet, un outil de simulation à évènements discrets a était conçue avec le but d'augmenter le spectre de vision des holons dans un HMS dans (Cardin and Castagna, 2009). Dans (Valckenaers et al., 1995) un système de contrôle pour cellule de production est présenté. Leur travail, proposant la création de pilotes de dispositifs basées sur une librairie d'opérations non-interruptibles, met en œuvre un ordonnanceur réactif offrant des plans de production de manière périodique et en ligne, basé sur un graphe dynamique des précédences avec des contraintes de processus et de capacité. Une série de tests de benchmark ont été menés pour comparer les performances des architectures hiérarchiques, hétérarchique et holoniques dans différents scénarios. Dans (Pach et al., 2014), un comportement holonique basculant d'un mode prédictif à un mode réactif est présenté. Cette approche bénéficie de l'avantage des méthodes prédictives pour générer des plans de production optimaux et bascule à un mode réactif lorsqu'une perturbation survient pour profiter alors de leur adaptabilité et pouvoir continuer la production sans besoin de l'interrompre. Une approche similaire, proposant un ré-ordonnancement dynamique d'opérations dans un HMS basculant aussi entre des stratégies de ré-ordonnancement rapide et des stratégies globales pour conduire leur optimisation.

5.2 LES APPROCHES IPPS : PLANIFICATION ET ORDONNANCEMENT INTEGRES

Traditionnellement, les fonctions de planification et d'ordonnancement ont des interactions ou collaborations très faibles, voire nulles. La fonction de planification continue à générer des plans de production en cherchant à trouver la meilleure manière de réaliser un produit, se basant uniquement sur les prérequis géométriques des composants ainsi que les contraintes technologiques des opérations. La PdP assigne des machines-ressource, des outils et des paramètres et une séquence à chaque opération en se basant sur des suppositions souvent irréalistes comme une complète disponibilité de toutes les ressources et matériaux.

Les systèmes CAPP, par exemple, ne regardent que vers le haut, visant une intégration hors-ligne avec les systèmes CAO. Généralement, ces systèmes négligent le potentiel d'une intégration vers le bas avec la fonction d'ordonnancement et ainsi ignorent l'effet que des conditions changeantes peuvent avoir sur la pertinence des plans processus (Tan and Khoshnevis, 2000). En effet, les changements qui peuvent apparaitre durant l'exécution des plans dus à des perturbations, comme l'arrivée ou l'indisponibilité des nouveaux produits et ressources, ne sont pas retournés à la fonction de planification. De plus, la création de plans de production fixes entraine très souvent des goulots d'étranglement, des déséquilibres de charges dans l'utilisation des machines et par conséquent une réduction de l'utilisation globale du système (Tan and Khoshnevis, 2000).

D'autre part, la fonction d'ordonnancement, elle aussi, trouve des complications lorsqu'un ré-ordonnancement devient nécessaire. Les plans fixes limitent la flexibilité pour trouver des solutions alternatives et/ou plus performantes par le moteur d'ordonnancement.

En vue de ces problématiques, cependant, l'intégration de l'ordonnancement avec d'autres fonctionnalités n'a pas reçu beaucoup d'attention. C'est depuis les années 90 qu'ont commencé les investigations sur les avantages à fusionner ces deux activités. Les résultats ont révélé qu'une telle intégration peut entrainer des améliorations significatives par rapport à une large gamme de mesure de rendement du système. Par exemple, (Chen and Khoshnevis, 1993; Palmer, 1994) ont rapporté que la fusion de ces deux activités apporte des améliorations significatives aux objectifs tels que la réduction du makespan, la réduction du retard moyen, la réduction de l'en-cours, l'augmentation de la probabilité de satisfaire les dates d'échéance et l'augmentation de l'utilisation des ressources. En outre, la séparation des deux activités impliquant la superposition ou duplication des efforts pour trouver des solutions, les deux activités doivent explorer leurs propres espaces de solutions séparément. Une intégration de ces deux

activités avec succès promet ainsi de réduire d'une manière très importante les efforts dans la recherche des solutions.

Le domaine de *Planification des Processus et Ordonnancement* (IPPS pour *Integrated Process Planning and Scheduling*) a pour objectif de réduire l'écart entre ces deux activités en créant des plans de production qui correspondent à l'état réel du système, considérant à la fois les contraintes présentes dans la planification et l'ordonnancement. Notamment, la conception des gammes non-linéaires (*Non-Linear Process Plans,* NLPP), aussi appelés plans de production flexibles, est devenue incontournable dans cette discipline par presque toutes les publications académiques (Shen et al., 2006).

Dans la section suivante, on présente une méthodologie pour créer et ordonnancer, de façon intégrale, des plans de production en se servant du modèle des gammes flexibles présenté dans le Chapitre 4. L'approche est proposée avec un protocole d'interactions holoniques au sein d'un *SoHMS* et avec des stratégies pour diriger la recherche des solutions de manière plus efficace vers une solution la meilleure possible.

5.3 ORCHESTRATION DE PROCESSUS-PRODUIT DANS UN *SOHMS*

5.3.1 Orchestration vs. Chorégraphie

Dû à la nature décomposée et distribuée des architectures SoA, l'aspect de la commande est principalement lié à la façon avec laquelle les services se mettent en relation pour travailler ensemble dans l'objectif l'ajout de valeur dans la formation de processus plus complexes. Comme défini par (Erl, 2005), les interactions entre les services sont généralement définis comme une séquence de procédures, appartenant la plupart du temps à différentes entités distribuées, lesquelles nécessitent d'être suivies pour accomplir un objectif commun.

A ce sujet, les approches et méthodologies développées pour la structuration des telles interactions, peuvent être classées en deux grandes approches : une *Orchestration* des services ou bien une *Chorégraphie* des services (ou même une combinaison des deux), Figure 59 .

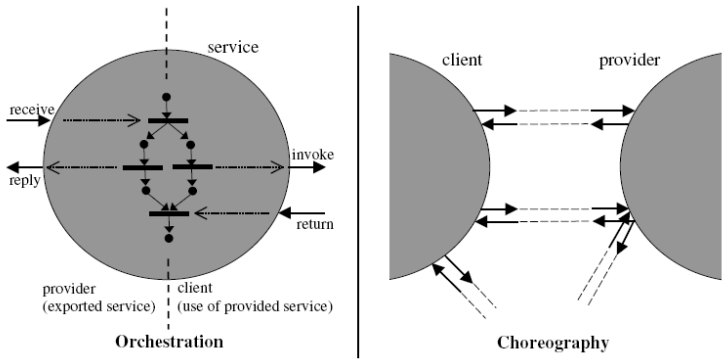


Figure 59 Principes d'Orchestration et de Chorégraphie (Pahl and Zhu, 2006)

Dans SoA, une *orchestration* est un ensemble de services coordonnés par une entité centrale. Cette entité centrale, appelé l'orchestrateur, coordonne l'exécution des services par rapport à une logique de contrôle, composée des règles, conditions et évènements. Suivant les principes de SoA, une orchestration peut être proposée comme un service en soi, les services n'ont donc pas besoin de porter de l'information sur la manière avec laquelle ils interagissent avec d'autres services ni sur leurs relations. Toutes ces informations sont comprises dans la logique de contrôle portée par l'orchestrateur. La structure d'une orchestration, étant hiérarchique, est extensible et composable et peut avoir une nature récursive. Comme un processus-produit (présenté Chapitre 2), une orchestration est une abstraction d'un processus et peut être

encapsulée comme un service. Une orchestration peut comprendre des services qui sont eux-mêmes d'autres orchestrations, d'où la nature récursive, Figure 60 .

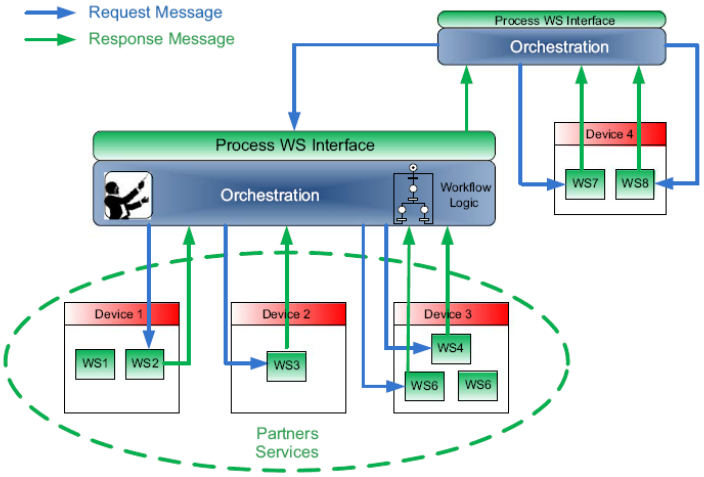


Figure 60 Orchestration Récursive (Cândido et al., 2009)

À l'opposé d'une orchestration, il n'existe aucun coordinateur désigné dans une *Chorégraphie*, Figure 61 . En revanche, la logique de contrôle n'appartient pas à une seule entité, mais à tous les participants de la chorégraphie. L'idée est de créer une collaboration organisée parmi les services distribués sans aucune entité centrale. De la même manière qu'un ensemble de danseurs synchronisés, les participants à une chorégraphie se coordonnent en percevant le comportement des autres participants. L'absence d'entité centrale implique le besoin des services participants d'être modélisés pour porter de l'information sur leurs rôles dans le processus global formé par la chorégraphie.

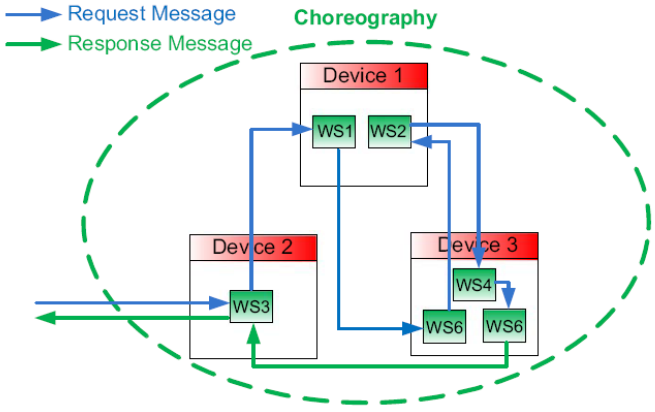


Figure 61 Chorégraphie (Cândido et al., 2009)

Les deux approches, orchestration et chorégraphie, peuvent coexister dans un même système. Comme on peut le voir dans la Figure 62 , à partir du moment où une orchestration a besoin de se coordonner avec une autre orchestration de même niveau, on commence à créer une chorégraphie d'orchestrations. Dans un *SoHMS*, les interactions peuvent être classées dans cette dernière catégorie. Les holons Produit, suivant les lignes directrices édictées par les systèmes contrôlés par le produit, deviennent les orchestrateurs du processus de production de chaque produit, ayant les gammes flexibles comme logique d'orchestration. Plus en détail, un *HP* coordonne l'exécution des services compris dans leur gamme de production en respectant les contraintes séquentielles indiquées par le RdP de la gamme. Ainsi, on construit une orchestration entre le représentant virtuel du produit et les ressources fournisseurs des services. On trouve alors une chorégraphie dans les interactions entre les *HPs* : plusieurs produits interagissant de manière simultanée, chacun cherchant la réalisation de son parcours de production, il

existe sans doute le besoin d'un mécanisme de coordination pour y parvenir en évitant des conflits d'utilisations de ressources de production ou de transport. Dans le *SoHMS*, les *HPs* forment une chorégraphie à travers un mécanisme de coordination indirecte basé sur la Stigmergie.

Cette manière mixte de structurer les interactions holoniques présente deux grands avantages :

- L'encapsulation de la logique de contrôle d'un processus dans une entité (l'orchestrateur : HP) facilite l'intégration des différents services dans plusieurs orchestrations, i.e. différents processus. Grâce à cela, les conception et modélisation des *MServices* deviennent plus simples, car l'on ne nécessite pas de modéliser les informations relatives à leur rôle dans les interactions, et préserve l'autonomie des *MServices* ignorant l'existence des autres services (Cândido et al., 2009).
- Leur caractère récursif correspond avec celui du produit et de ressources dans un *SoHMS*. La relation {Produits/Sous-produits} est analogue à la relation {HP/MService} (les *MServices* pouvant représenter l'intégration d'un sous-produit au produit de base) et à la relation {*MServices* Composés/MServices} où l'exécution d'un service composé peut être réalisée à travers une orchestration réalisée par le holon de la ressource composée.

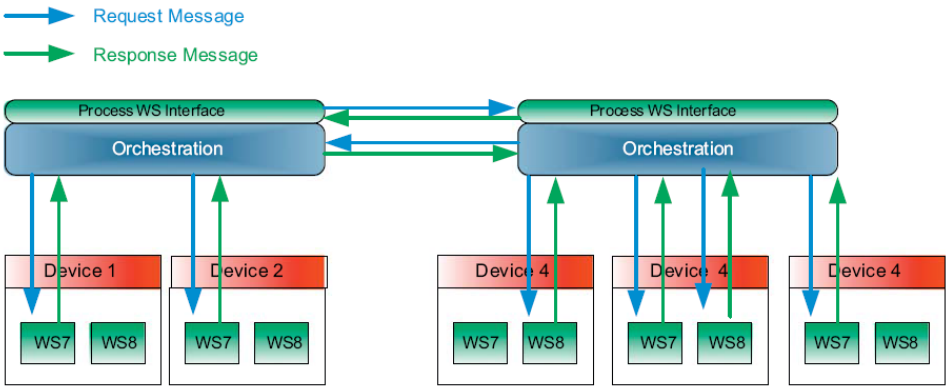


Figure 62 Coexistence entre Orchestration et Chorégraphie (Cândido et al., 2009)

5.3.2 Orchestration des plans de production

Dans un contexte IPPS, le problème d'ordonnancement, originalement perçue comme un problème de deux dimensions (machine, temps), devient alors un problème en trois dimensions (machine, temps, méthode). Dans une application *SoHMS*, on peut formuler ce nouveau problème de la manière suivante :

> *Etant donné un ensemble de k produits composés par n services et un ensemble de m machines (avec des méthodes alternatives), l'ordonnancement dans une architecture SoHMS cherche à trouver une séquence de services pour chaque produit satisfaisant les contraintes de production inscrites dans leur gamme de fabrication, et simultanément une séquence de machines capables de satisfaire la séquence de services et satisfaisant les contraintes de disponibilité actuelle des ressources dans l'atelier, qui soient les plus performantes calculées dans une plage de temps de x.*

Ne pas seulement sélectionner une séquence de machines pour la réalisation d'un produit mais également l'ordre des opérations pour un produit augmente naturellement la complexité du problème. De plus, dû au besoin de réactivité du système, cette recherche doit être capable de renvoyer des solutions acceptables dans un délai relativement court par rapport aux méthodes traditionnelles. Cette fenêtre de temps est définie par la différence entre le temps présent et la date d'arrivée au prochain point de prise de décision.

Plusieurs termes doivent être définis ici avant de poursuivre dans notre proposition de protocole d'orchestration. Au sujet des espaces de solutions:

- **Espace de solutions séquentielles**: Cet espace de solutions est défini par la nature des processus et les opérations qui le forment. Il est délimité par les contraintes temporelles et séquentielles des opérations ainsi que les relations de dépendance entre elles. Cet espace de solutions est indépendant de tout atelier et est porté par le modèle *SoP* de gamme flexible d'un produit.
- **Espace de solutions faisables** : Ceci est un espace de solutions ayant comme axes l'espace de solutions séquentielles et les capacités de l'atelier où le produit sera fabriqué. Les solutions dans cet espace comprennent une séquence d'opérations satisfaisant les contraintes imposées par le modèle *SoP* et une route d'acheminement au sein de l'atelier qui compte avec les capabilités nécessaires pour réaliser les opérations demandées. Les solutions sont faisables sous des conditions idéales du système, ne prenant pas en compte par exemple la disponibilité des ressources, i.e. l'axe temporel.
- **Espace de solutions réelles** : Cet espace ajoute l'axe temporel à l'espace de solutions faisables. Il représente, pour une durée déterminée (jusqu'à l'arrivée d'un nouvel évènement), l'ensemble des solutions valides avec lesquelles on peut fabriquer un produit. Une solution comprend alors une séquence d'opérations valides, une route d'acheminement réalisant cette séquence et un emploi du temps pour chaque opération plaçant le parcours de production d'un produit dans le système de production.

Au sujet des solutions :

- **Séquence Faisable (SF)** : collection d'instances de *MServices* (Spécification de MService) inclus dans le modèle de produit et respectant les contraintes de séquencement imposées indiquées par la gamme du produit. Appartient à l'espace de solutions séquentielles.
- **Route Faisable (RF)**: représente un chemin physique d'acheminement dans le système de production qui satisfait une SF, les contraintes d'atteignabilité et les besoins de capabilités de chaque service (i.e., chaque service est attribué à une ressource avec les capabilités nécessaires et il existe une séquence de services de transport entre les machines de deux services successifs).
- **Plan Faisable (PF)** *:* RF distribuée dans un axe de temps. Un PF est une solution complètement déterminée et valide pour son exécution dans un atelier prenant en compte l'état d'utilisation du système à jour. Tout le parcours d'un produit est indiqué : machines, services, chemin de transport entre machines, spécification des *MServices* et leurs paramètres et le temps de fin et de début de chaque action du plan.

Notre approche présente un protocole d'orchestration basé en grande partie sur le protocole d'interactions Contract-Net (CN-P) de FIPA (Foundation for Intelligent Physical Agents) et l'approche de prévisions à court terme présentée dans (Belle et al., 2009; Holvoet and Valckenaers, 2006; Valckenaers et al., 2006) inspiré par l'organisation des colonies de fourmis. Le protocole CN-P est utilisé pour régir et coordonner l'échange d'informations dans les interactions de négociations entre les Holons Produit (clients) et les Holons Ressources (fournisseurs). Les principes de l'approche de vision à court terme sont utilisés pour réduire le comportement myopique (Adam et al., 2011; Zambrano Rey et al., 2014, 2013), caractéristique des systèmes distribués, et rendre aux participants une vision globale du système pour la durée de son cycle de vie.

Comme mentionné précédemment, cette approche reprend certains concepts de travaux présentées par (Belle et al., 2009; Holvoet and Valckenaers, 2006; Valckenaers et al., 2006) pour l'ordonnancement des tâches avec une vision à court terme, notamment l'utilisation d'agents appelés agents fourmis ou agents délégués. Ces agents fourmis ont des activités relatives à la requête d'informations du système et à la

dissémination des plans au travers du système. Leurs principaux objectifs sont la découverte des capabilités de production, l'exploration des capacités du système dans son état réel et la réservation et communication des plans de production. Il existe alors trois types d'agents fourmis : Les *Fourmis de Faisabilité (FdF),* les *Fourmis d'Exploration (FdE),* et les *Fourmis d'Intention (FdI)*, chacun avec un comportement différent dont l'ensemble des efforts donnera à son créateur, le *Holon Produit*, une vision virtuelle de l'état futur du système pour ainsi trouver une solution de production et s'inscrire dans l'activité future de ce système. Plus avant dans ce document, on présentera comment ces agents fourmis entrent en scène dans le processus d'orchestration avec une description plus détaillée de leur comportement.

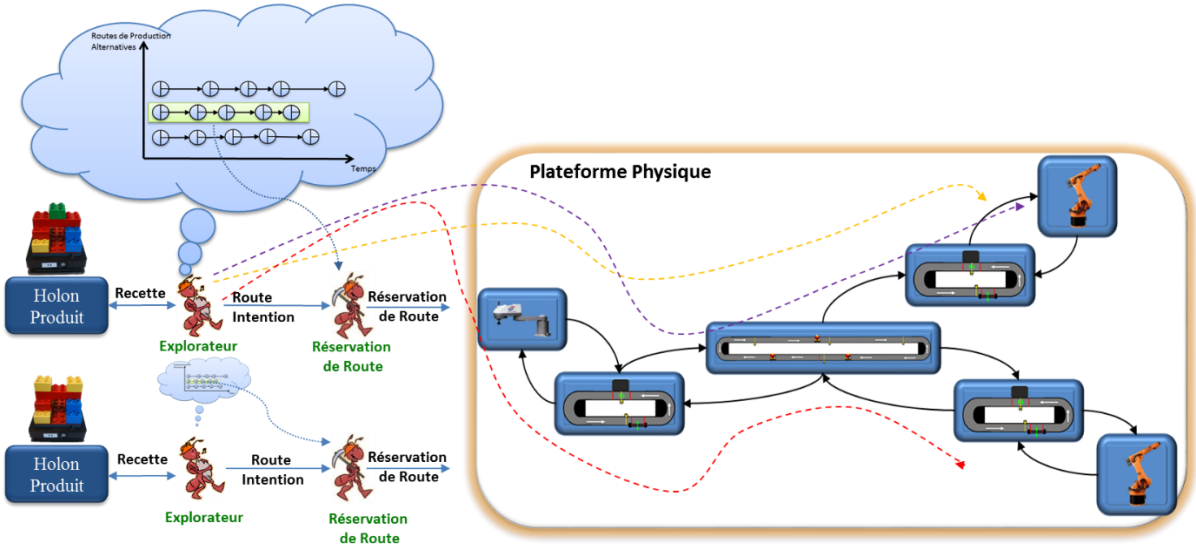


Figure 63 Orchestration basée sur l'approche des Colonies de Fourmis

Le processus d'orchestration de notre approche se divise en cinq grandes phases : une *Phase de Faisabilité* pour la construction d'un espace de solutions faisables, une *Phase d'Exploration* pour l'exploration du plan d'utilisation des ressources à l'état actuel du système, une *Phase d'Intention* pour l' évaluation et la sélection d'une solution de production et leur dissémination dans les plans de production globaux ; une *Phase d'Exécution* en charge de commander et surveiller la bonne exécution du plan de production choisi ; et une *Phase Réactive* qui entrera en jeu lorsque le système détectera une perturbation dans le système ou une solution alternative plus performante. Voici une description des actions de chacune de ces phases :

Phase de faisabilité

Cette première étape du processus d'orchestration consiste en la construction d'un espace de solutions faisables au travers de la découverte des capacités du système physique de production. Le *Holon Produit*, en se servant de son propre modèle flexible de gamme, se lance dans la construction d'un *graphe dynamique de faisabilité* dont on peut attendre des trajectoires faisables respectant les contraintes de séquence, d'accessibilité et devant répondre aux besoins de transformations du produit selon la configuration de l'outil de production.

Pour la construction d'un tel graphe dynamique, le *Holon Produit* crée deux groupes de fourmis de faisabilité. Ces fourmis de faisabilité ont alors deux activités principales : tout d'abord, l'exploration de l'espace de solutions séquentielles et ensuite la découverte des capabilités de routage et transformation des ressources présentées au sein du système. Les solutions trouvées dans cette étape correspondent à des routes faisables. Pour ceci, les FdF parcourent virtuellement le système à partir de la position du HP et avancent-en suivant les trajectoires qui satisfont les contraintes imposées par la gamme du produit et l'accessibilité des ressources. Dû à l'objectif de réactivité du système, deux défis se présentent dans cette

étape : la découverte de solutions complètes et valides et la découverte des solutions les plus prometteuses pour des conditions idéales. Le succès du premier contribuera à la réactivité pour permettre de proposer un ensemble de solutions valides au HP avant de rencontrer un point de décision, tandis que le succès du second contribuera à la performance du système en permettant l'exploration des solutions les plus performantes. Pour atteindre ceci, les FdFs sont créées avec deux comportements de recherche différents : un comportement d'exploration arborescente avec l'algorithme *Best-First Tree Search* (*FdF de profondeur*) et un comportement d'exploration transversale de graphe avec l'algorithme *Best-First Transversal Search (FdF transversal),* pour une description détaillée de ces algorithmes se référer à (Dechter and Pearl, 1985; Hart et al., 1968; Pearl, 1984) La recherche en profondeur répond au premier défi, car les FdF trouvent très rapidement des routes faisables en suivant les branchements les moins coûteux selon le critère d'évaluation (pour le graphe dynamique de faisabilité, on se réfère au temps de traitement nominal pour les opérations de transport et de transformation). Le FdF transversal donne des solutions à ce problème en se concentrant toujours dans l'exploration de la trajectoire la moins coûteuse mais cette fois-ci en largeur du graphe exploré. Ce comportement, avec une vision d'exploration plus large, garantit de trouver la trajectoire représentant le minimum global, mais dans une période de temps plus grande. Le lancement de FdFs avec ces deux comportements permet alors de trouver des solutions rapidement sans toutefois abandonner l'idée d'exploration ciblée de meilleures solutions.

Le graphe dynamique de faisabilité proposé, Figure 64 , est un automate fini décrit par $GF = \{Q, T, \partial, \Gamma, q_0\}$ où :

- Q est un ensemble fini d'états, chaque état défini par $q = \{hr, ms, port, H\}$ où :
  - *hr* est un *Holon Ressource* capable d'exécuter le *MService ms* arrivant par son *port.*
  - H est un ensemble des « historiques» ou chaque « historique » est défini par $hist = \{m, g(hr,ms,m)\}$ où :
    - *m* est une valeur de marquage, issue du RdP de la gamme du produit, correspondant à un état de production du produit avant l'exécution de *ms.*
    - *g(hr,ms,m)* est le coût dépensé dès l'état initial *qo* jusqu'à *q,* en fonction de sa ressource, son service et le marquage de la gamme.
- *T* est un ensemble fini de Transitions : chaque transition représentant un *Meta-Transport* [6] défini comme une séquence de services de transports effectués par différentes ressources.
- q_0 est l'état initial du graphe, localisant le produit au port d'entrée p_0 et au marquage initial m_0.
- ∂ est la fonction de transition : $Q \times T \rightarrow Q$
- $\Gamma : Q \rightarrow 2^T$ est la fonction des transitions sortantes : σ *(hr,ms,port,m)* est l'ensemble des transitions habilitées dans un état en fonction du marquage de son H.

[6] Ce terme fait référence au terme présenté dans les travaux de (Bourdeaud'huy and Toguyeni, 2006; Toguyeni, 2006), où il est utilisé le terme *Meta-Transfert.*

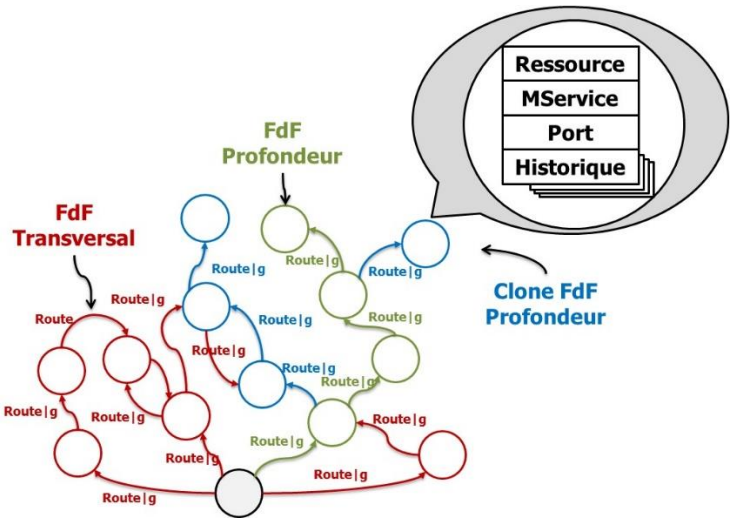


Figure 64 Structure du graphe dynamique de faisabilité

Pour construire ce graphe de faisabilité de manière dynamique, le HP créé et lance un ensemble de FdFs de profondeur et un seul FdF transversal à partir de la position du produit et de l'état de son cycle de vie lié à la gamme. Les pas de cette première phase sont décrits ci-dessous :

Comportement des FdFs de Profondeur :

1. *Calcul des Services et des marquages*; le FdF demande au HP l'ensemble des MServices habilités selon la gamme et l'état de production du produit. Le HP retourne une liste avec la couple: *(service | marquage).*
2. *Découverte des Fournisseurs*; Pour chaque MService dans la liste, la FdF envoie une requête de service au DF pour lui demander toutes les ressources déclarées dans le système capables d'exécuter les MServices. Le DF explore ses registres et retourne une liste des arcs comprenant les HR, les Profils de MService associés et la valeur de son coût nominal.
3. *Découverte des Routes*; inclus dans l'appel de service précédent, le DF retourne une route de transports pour arriver à la ressource indiquée (y compris son port d'entrée), accompagnée également de son coût nominal. Dans les systèmes multi-routages, un HR peut apparaitre dans plusieurs arcs dû aux différentes alternatives de routage découvertes par les algorithmes implémentés par le DF7 .
4. *Création des Arcs*; La FdF construit alors un ensemble d'états *(HR|MService|Port|H)* liés à un méta-transport représenté par une transition; les deux forment alors un arc du graphe. Le FdF ajoute l'ensemble des arcs au graphe dynamique de faisabilité.
5. *Calcul des historiques*; pour tout état créé et chaque marquage créé, la FdF calcule la valeur du coût de chaque historique. Partant de l'état initial, le coût d'une méta-transport est calculé comme la somme des coûts nominaux de chaque service $g(t)= g(st1)+ g(st2)+ g(st3)+..$ tandis que le coût d'un état est le coût nominal indiqué par le DF, fonction de la ressource et du MService (dans ce cas le temps nominal de traitement $g(q)=g(hr,ms)=Cmax$). Donc, pour chaque historique, le coût nominal est la somme des coûts de tous les méta-transports plus le cout de traitement de tous les états de la plus courte route entre l'état initial et l'état après l'exécution du service de tel état : $g(hr,ms,m)= g(t)+ (g(hr',ms',m')= g(t')+g(hr'',ms'',m''))$

[7] Le Répertoire Facilitateur (DF), étant le constructeur et le porteur du plan de configuration de l'atelier, offre des services de calcul de trajectoires de transport entre deux ports du système de production. Les algorithmes implémentés, sont du type plug-and-play et peuvent être sujets à calibration selon les intérêts de l'application. Par exemple, pour des applications avec un grand nombre des trajectoires, il peut être pertinent de limiter la proposition de solutions à un nombre limité pour ainsi éviter une explosion inutile du nombre de solutions.

6. *Sélection et découverte d'une route faisable*; Parmi tous les historiques, la FdF choisit la solution la moins coûteuse (état avec un marquage) et se déplace virtuellement à la prochaine étape en marquant l'historique comme parcouru.
7. Les pas 1-6 sont répétés jusqu'à ce que la FdF n'aboutisse à une solution complète. Si un état se répète dans l'exploration, on ajoute un nouvel historique et on continue la construction de l'historique. Si la FdF tombe dans un état puits, i.e. un état où on ne peut plus évoluer (normalement caractérisé dans les systèmes de production comme des points d'inter-blocages), dans ce cas la FdF revient en arrière en marquant l'historique comme non-faisable et essaye avec un autre arc. Cette procédure de retour en arrière se répète jusqu'à que on aboutisse à une solution complète réalisant tous les MServices indiqués dans la gamme du produit.
8. Ayant une solution complète, la FdF parcourt le chemin en arrière en marquant les historiques comme explorés et en collectant la séquence des états valorisée par son coût vers l'état objectif de la route découverte. Ce dernier servira comme base à l'heuristique d'exploration dans la phase d'exploration.
9. La FdF envoie la route au HP pour la construction d'un arbre de solutions faisables (ou arbre d'exploration).
10. Suite à la découverte de cette route, la FdF continue l'exploration des branches au long de la route. De manière aléatoire, la FdF revient à un point intermédiaire de la route et clone un FdF de profondeur pour explorer une branche non explorée à partir du point choisi. Le clone, à son tour, répètera le comportement d'exploration pour la branche et pour l'exploration des branches successives sur la nouvelle route. Le taux de clonage peut être proportionnel au nombre des fils de « threads » (fils d'exécution). Ainsi, à chaque génération de clonage, le nombre de clones diminue jusqu'à un plafond de nombre maximal de FdF simultanées. Ce paramètre nécessite une calibration par rapport aux capacités de calcul des processeurs utilisés. Un plafond bas diminue le calcul parallèle se concentrant sur l'exploration des solutions complètes plus rapidement. Un plafond haut diminue la probabilité de rester dans un minimum local donnant des solutions plus variées. De plus, la combinaison des actions de clonage et la sélection aléatoire favorise l'exploration hors de son minimum local. Dans la Figure 65 on peut observer comment, à mesure que le nombre de FdF se multiplie, on augmente la probabilité de choisir des branchements plus près de la racine de la route, donc augmenter la probabilité d'explorer des trajectoires complètement alternatives. Plus important encore, cet incrément de priorité à l'exploration des étapes plus près de la racine va en complet accord avec les objectifs d'exploration: donner priorité à l'exploration des alternatives immédiates permet d'avoir une vision plus générale de l'espace de solutions et permet de se diriger vers le minimum global.

Comportement de la FdF Transversale :

1. Les pas 1,2 et 3 du comportement du FdF de profondeur sont répétés.
2. Le FdF ajoute dynamiquement l'ensemble des arcs au graphe de faisabilité, il choisit parmi une liste de tous les arcs des solutions découvertes et non explorées, i.e. celui qui représente la solution la moins coûteuse jusqu'au point présent d'exploration. Toutes les FdFs (de profondeur et transversales) travaillent sur le même graphe de faisabilité. De cette façon la FdF transversale se sert des explorations faites en profondeur évitant la duplication des efforts.
3. La procédure se répète jusqu'à ce que le FdF ne trouve la solution du minimum global selon les valeurs nominales et l'hypothèse d'une disponibilité complète. La FdF parcourt alors le chemin en arrière, de même que le font les FdF de profondeur, en laissant une marque d'exploration et calculant la valeur du coût vers l'état objectif pour la phase d'exploration.

4. La FdF envoie la route au HP pour la construction d'un arbre de solutions faisables.
5. La FdF transversale continue sa recherche de la deuxième (prochaine) solution plus performante. L'agent fourmi reprend son exploration pour les historiques non explorés jusqu'à la découverte de la seconde route la moins coûteuse. Cette procédure continue tout au long du cycle de vie jusqu'à la découverte totale de l'espace de solutions faisables.

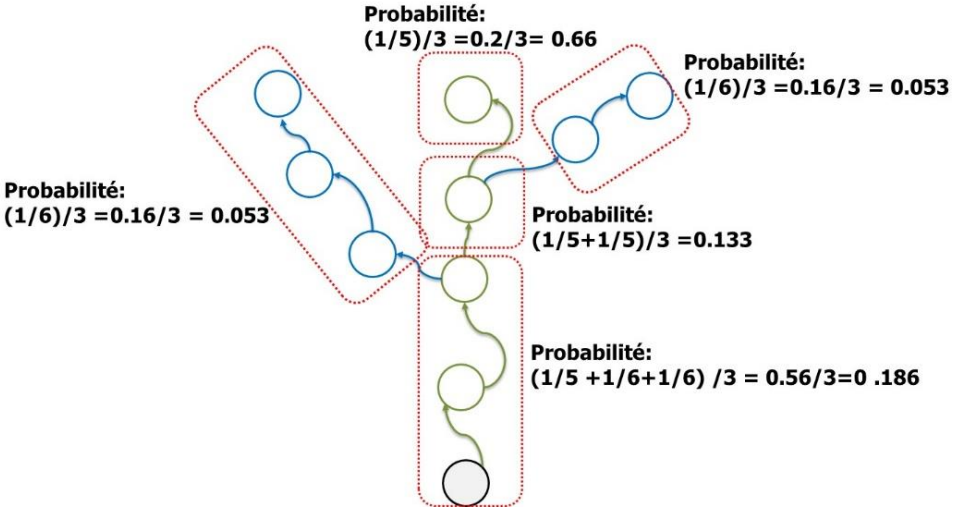


Figure 65 Probabilité d'exploration de chaque place dans le Graphe de Faisabilité (trois fourmis actives)

Phase d'exploration

Cette deuxième étape consiste en l'exploration de l'espace des solutions Réelles du système, en récupérant les informations sur l'état du système et le plan d'utilisation de ses ressources. Cette exploration est déléguée aux Fourmis d'Exploration (FdE), lesquelles à travers le protocole d'interactions Contract-Net font appel à des propositions des *HRs*, communicant ainsi leur utilisation. L'objectif de cette phase, comme pour la phase de faisabilité, est de calculer des solutions complètes pour le plan de production du produit dans des délais de temps acceptables, déterminés par les fenêtres de temps entre les points de décision dans la trajectoire initiale. Pour ce motif, l'exploration emploie une stratégie basée sur une heuristique avec deux comportements des FdEs.

L'exploration se fait sur un graphe de solutions faisables (ou arbre d'exploration), construit par le HP à partir des découvertes des fourmis de faisabilité pendant la phase de faisabilité. Cette arbre, Figure 66 , est composé d'un automate fini défini par $AdE = \{Q, T, \partial, q_0\}$ où :

- Q est un ensemble fini d'états : chaque état défini par $q = \{hr, ms, port\}$ où *hr* est un *Holon Ressource* capable d'exécuter le *MService ms* arrivant par son *port*.
- T est un ensemble fini de Transitions, comme dans le graphe dynamique de faisabilité.
- q_0 est l'état initial du graphe localisant le produit au port d'entrée p_0.
- ∂ est la fonction de transition : $QxT \rightarrow Q$
- Toute trajectoire dans cet arbre représente une solution dont il existe tous les capabilités pour son exécution au sein de l'atelier. C'est avec cet arbre que l'on lancera des agents fourmis pour l'exploration des disponibilités des ressources, et pouvoir ainsi parvenir à la sélection d'une solution intention. Cet arbre des solutions est construit dynamiquement tout au long de la phase de faisabilité.

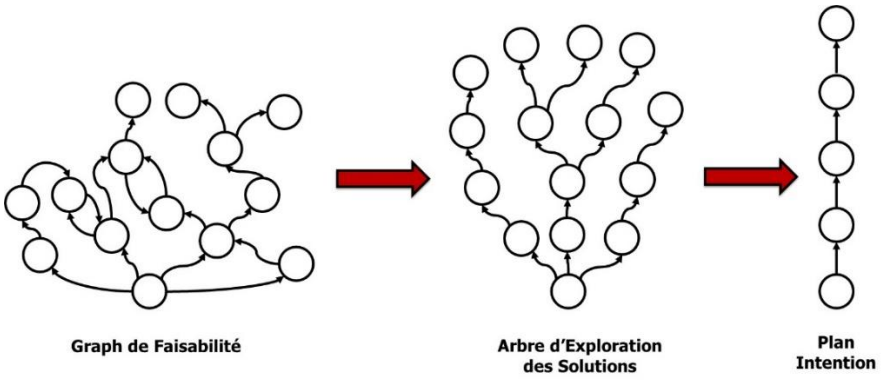


Figure 66 Les graphes d'espaces de solutions

Comme lors de la phase de faisabilité, deux types de comportement sont utilisés : un comportement d'exploration basé sur l'algorithme *A* (FdE Transversale)* pour la recherche de la solution optimale et un comportement d'exploration en profondeur avec l'algorithme *Best-First Tree Search* (*FdE de profondeur*) avec l'agrégation d'une heuristique ayant fonction d'évaluation tel que défini pour l'algorithme A*. Leurs fonctions sont analogues à celles des FdFs de profondeur et FdF Transversales : les FdEs de profondeur s'occupent de trouver des solutions rapidement tandis que les FdEs Transversales s'occupent de trouver la solution optimale (avec une perspective locale au HP et par rapport aux informations récupérées). Leur exploration est dirigée par la fonction d'évaluation $f(a) = g(a) + h(a)$ où $g(a)$ est le coût du point du départ jusqu'après l'exécution de l'arc *a*, et $h(a)$ est la fonction heuristique, dans ce cas, le meilleur *coût nominal* après l'exécution de l'arc *a* jusqu'à la fin d'une route faisable, calculée lors de la phase de faisabilité par les FdF. Selon la théorie des graphes, le protocole A* est considéré optimal (i.e. l'algorithme trouve la trajectoire la moins coûteuse) lorsque la fonction heuristique est admissible. Pour qu'une fonction heuristique soit admissible, elle ne doit pas surestimer le coût réel des solutions, c'est-à-dire que la valeur de h(n) ne doit jamais être estimée comme inférieure à la valeur réelle. Du fait que notre heuristique est déterminée par le temps d'exécution de chacune des activités le long d'une route de production, elle représente un plan idéal sans périodes d'attente avec une disponibilité totale des ressources utilisées. Par conséquent, cette heuristique est admissible car elle représente des solutions idéales pour chacune de ces routes. En outre, l'algorithme A* est dit efficacement optimal pour une heuristique, i.e. il n'existe autre algorithme de recherche qui garantisse trouver la solution optimal en déployant moins de nœuds que A*.

L'étape d'exploration commence dès qu'une solution complète a été découverte par l'une des FdFs. Alors, les deux étapes (exploration de la faisabilité et exploration de la disponibilité des ressources) se font de manière parallèle. Ceci prend du sens en rappelant l'objectif de réactivité : lorsqu'une solution est découverte, on se lance à son exploration pour rapidement avoir une solution réelle de production. L'exploration se fait alors sur un arbre de solutions qui se construit de manière dynamique et parallèle. Comme pour la phase de faisabilité, le HP lance un groupe des FdEs de profondeur et une FdE transversale. Voici les étapes décrivant cette phase d'exploration :

Comportement des FdEs de Profondeur :

1. La FdE se place sur l'arbre des solutions faisables créé par les fourmis de faisabilité, sur l'état actuel du HP.
2. Pour chaque arc disponible, la FdE s'engage avec le CN-P à la recherche de propositions de service de la part des ressources. Le protocole commence avec un *message d'appel à proposition de service,* indiquant la spécification de service nécessaire ainsi que la date d'arrivée au port d'entrée de la ressource. Ensuite, le HR à son tour évalue l'appel en fonction de son plan d'utilisation et son plan de configuration et retourne un message avec une *proposition de service.*
3. Cette procédure commence, d'abord, par l'exploration des services de transport spécifiés dans la transition des arcs. A chaque étape, la FdE collecte la date de fin de transport entre ports

consécutifs ; la FdE reçoit une date de fin de transport et envoie cette date comme date d'arrivée pour le service suivant et ainsi de suite.

4. Une fois explorée la disponibilité des ressources de transport et ayant une date d'arrivée au port de la ressource de production, de la même manière la FdE fait un appel de proposition de service pour le *MService* indiqué dans le graphe de faisabilité.
5. Une fois tous les arcs explorés, la FdE évalue tous les arcs suivant la fonction d'évaluation $F(a)= G_f(a) + h(a)$ où :
    - $G_f(a)= G_f(ms) \leq G_f(t)+Cmax(ms)+Smax(HR)$, qui représente la date de fin d'exécution d'un arc. Ceci est équivalent à la date de fin d'exécution du *MService,* qui est inférieure ou égale à la date de fin d'exécution des services de transport ($G_f(T)= G_f(t_1)+ G_f(t_2)+ G_f(t_3)+..$) plus le temps d'exécution nominale du *MService* (*Cmax*) plus le temps de réglage maximal de la ressource (*Smax*) ;
    - et h(a) est le temps d'exécution optimal jusqu'à la fin du cycle de vie de production calculé pendant la phase de faisabilité par le retour des FdFs.
6. Les FdE sélectionnent l'arc minimisant *F(a)* et se déplacent virtuellement à la fin de cet arc et faire progresser de manière virtuelle son horloge. La FdE se déplace alors vers un état futur de sa production (selon le plan global de disponibilité) et continue la recherche en profondeur, considérant uniquement les arcs sortant de ce nouvel état. La suite se déroule comme pour les FdFs de profondeur jusqu'à qu'on aboutisse à l'exploration d'une route complète dans le temps imparti, qui s'ajoute alors à la projection de l'état futur du système.
7. Ayant une solution complète, la FdE parcourt le chemin en arrière, annotant la valeur de *H(e),* qui est la date de fin de service la plus courte atteignable depuis cet état. Ce dernier servira pour la sélection de la route la plus performante.
8. Comme dans la phase de faisabilité, suite à la découverte de cette route, la FdE continue l'exploration des branches le long de la route de manière aléatoire en créant des clones avec un taux de création en fonction du nombre des FdEs clonées.

Comportement des FdEs Transversales:

Le comportement des FdEs transversales correspond à celui de l'algorithme de recherche A* avec en plus une étape d'appels à propositions de service, déjà présentée ci-dessus. A la différence des FdEs de profondeur, la recherche ne comporte pas seulement une route mais plusieurs simultanément en fonction de celle qui évolue de la façon la plus prometteuse. A chaque étape, tous les arcs non explorés sont sélectionnés en fonction de la fonction d'évaluation F(a) représentant la date de fin d'exécution estimée avec un composant réel (G(a)) et un compostant estimé (h(a)). A la fin de son exploration, la FdE trouvera la solution optimale au problème parmi les routes du graphe d'exploration. Lorsque la FdE trouve une solution, elle se met en vieille jusqu'à l'apparition d'une nouvelle route à explorer plus prometteuse que la solution trouvé, $f(r)<F(r)$.

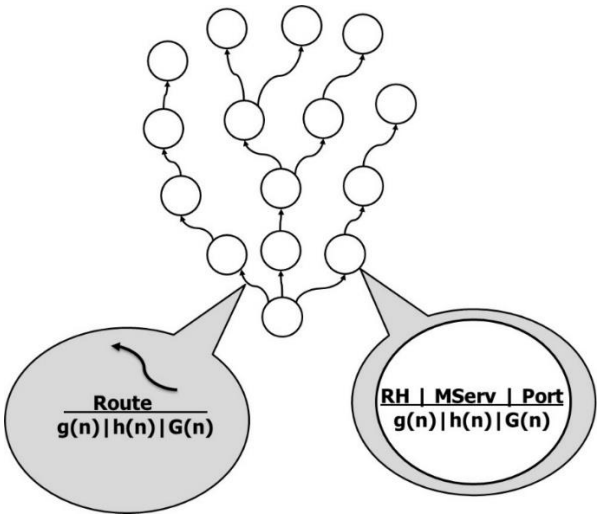


Figure 67 Champs d'informations des Places et Transitions

Phase d'intention

Cette phase consiste à la sélection d'une solution et son inscription dans le plan de solutions du système. Une fois que les efforts parallèles des FdF et des FdE ont trouvé des solutions complètes, faisables et valides, le HP peut définir un plan de production. On peut diviser cette phase en quatre étapes : une *étape d'évaluation*, une *étape de décision*, une *étape de contractualisation* et *une étape de confirmation*.

L'étape d'évaluation se fait lorsque la phase d'exploration retourne des résultats. Chacune des solutions trouvées sont classées dans une liste selon la valeur de leur fonction F(r). Cette étape conclut dans deux situations : soit on a trouvé la solution optimale du problème (s'il n'est pas de grande taille), soit on arrive à une échéance temporelle définie. Pour une exploration initiale avant lancement du produit, cette échéance est déterminée manuellement et est sujette à paramétrage. Si l'exploration se fait pendant l'exécution, alors une telle échéance est déterminée par le temps entre l'état présent et le prochain point de décision (la date de conclusion du service en exécution, soit du transport soit de transformation).

L'étape *de décision* commence alors avec une plage de temps, aussi à paramétrer, avant échéance. Cette plage doit être suffisante pour effectuer les interactions de l'étape de contractualisation. Dans cette étape, on sélectionne la solution minimisant le coût (le temps de production réel). Après la sélection commence l'*étape de contractualisation* avec la création d'une Fourmi d'Intention (FdI). Son comportement se décrit par les étapes suivantes :

1. La FdI se place à la position du HP dans l'arbre d'exploration.
2. La FdI, suivant les valeurs de H(a) des arcs, se déplace à travers l'arbre des solutions.
3. A chaque état, la FdI commence donne continuation au protocole Contract Net avec l'acceptation de la proposition et la constatation du service.
4. Le HR, à son tour, enregistre cette intention dans son plan d'utilisation pour ainsi gérer les futures propositions des autres produits et éviter les conflits d'intentions.
5. La FdI continue en avançant dans l'arbre jusqu'à la contractualisation de tous les services de la route. Dans le cas où une contractualisation ne peut être faite dû au changement de l'état futur du système, alors la FdI regarde une solution alternative à partir du dernier état réservé en suivant la prochaine H (a) la plus performante. Si la performance est très grande par rapport à la deuxième solution globale, alors il essaie de réserver la deuxième solution et envoie des notifications d'annulation aux *HRs*.

Phase d'exécution

Une fois avec un plan de production défini et réservé, on passe à la phase d'exécution du plan de production. Comme présenté dans le Chapitre 3, c'est le *Holon Ordre* (HO) qui s'occupe de cette tâche. Lorsqu'un plan est fixé, le HP lance un HO pour invoquer l'exécution des services et surveiller leur bonne exécution et le bon suivi du plan. Les invocations de service sont faites pour tout service (soit de transport ou MService) avec le protocole d'invocation présenté dans une section précédente.

Phase de réactivité

Une fois entré dans la phase d'exécution, le HP entre aussi dans la phase de réactivité. Pendant cette phase, les activités des phases de faisabilité, d'exploration et d'intention continuent avec certaines modifications. L'exploration de la faisabilité continue avec le même comportement, explorant des routes à partir d'un point initial, de l'état du produit. De ce fait, à chaque exécution d'un service, on transfère l'état du produit dans le graphe de faisabilité. D'autre part, l'exploration de l'arbre des solutions change légèrement son comportement d'exploration. Ces modifications se font sous l'hypothèse de la criticité du respect des contrats faits. Comme dans les organisations humaines, l'annulation d'un rendez-vous programmé dans l'avenir proche a un plus grand impact sur l'environnement par rapport à une annulation d'un rendez-vous programmé plus loin dans le futur. Cet impact se traduit normalement en un plus grand nombre d'activités correctives pour arriver à rattraper un plan de production. De ce fait, l'exploration se centre autour de la route d'intention. La FdE transversale continue avec une exploration normale. Différemment à la phase d'exploration, dans cette phase on lance au départ une seule FdE pour explorer le branchement de la route d'intention en clonant des FdEs de profondeur. Cependant, cette exploration se fait à différentes zones de probabilité. En effet, la route d'intention est divisée en trois zones, Figure 68 .

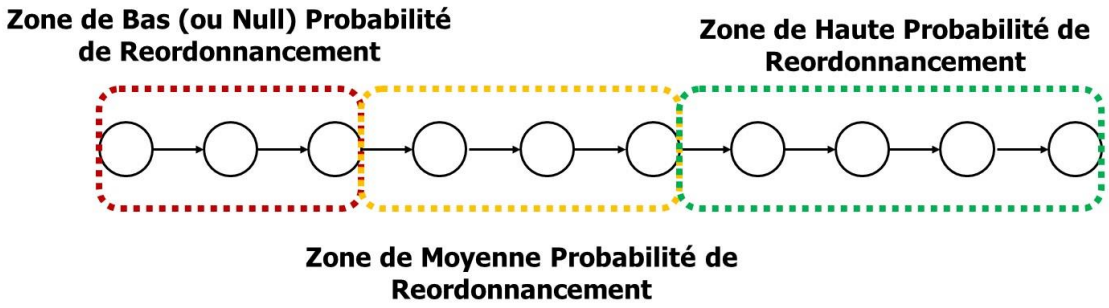


Figure 68 Zones de probabilité pour le changement de plans

La zone « immédiate » du plan de production est considérée d'une grande criticité et donc on lui attribue une probabilité d'exploration faible. La partie centrale est explorée avec une probabilité intermédiaire et l'étape finale (la plus lointaine dans le futur) est explorée avec une grande probabilité. Cette distribution est aussi appliquée pour la décision de changement des plans. Dans cette phase également, le HP lance une FdI. Ses objectifs dans cette phase sont la confirmation du plan de production établi et la prise de décision pour un éventuel changement de plans si une solution plus performante est découverte. La FdI, de manière périodique, parcourt le graphe des solutions à partir de l'état présent du produit. Dans des conditions non changeantes, la FdI suivra la route d'intention et enverra des confirmations aux *HRs* pour renforcer les plans chez les ressources. S'il se trouve que les FdEs trouvent une solution plus performante, alors la FdI l'identifiera et décidera s'il est nécessaire de changer de plan.

Lorsqu'on exécute un *MService,* toutes les fourmis avancent dans le graphe de faisabilité comme dans l'arbre des solutions. Cela permet de concentrer les efforts d'exploration vers des solutions utiles : à

chaque pas exécuté, l'espace de solutions se réduit aussi, permettant de rendre agile l'exploration pour ainsi trouver des meilleures solutions.

5.4 STRATEGIES D'EXPLORATION DE L'ESPACE DE SOLUTIONS

Comme décrit dans la section précédente, la complexité combinatoire est l'une des plus grandes limitation dans l'implémentation des gammes flexibles de production du fait qu'elles augmentent naturellement le nombre des solutions alternatives à explorer. De plus, ce nombre incrémente rapidement avec la taille du processus et la flexibilité structurelle de l'atelier, comme présenté dans la section 4.5.2. Enfin, il se trouve que beaucoup des solutions valides donnent des mesures de performance loin d'être optimales ou satisfaisantes. Ces solutions se caractérisent souvent par un très grand nombre de changements de séries sur machines, de sauts entre ressources provocant une augmentation du temps de transport et une complexification des flux de produits. Dans cette section, nous proposons des stratégies à intégrer dans le processus d'orchestration qui aideront à attaquer le problème combinatoire et permettront de cibler l'exploration vers un espace de solutions idéalement plus performant. Ces stratégies sont inspirées des approches proposées dans les projets FLEXPLAN et COMPLAN pour réduire la complexité combinatoire et augmenter le temps de réponse pour la génération de modèles de plans de production non-linéaires au travers du regroupement d'opérations similaires et le regroupement des composants ayant des caractéristiques aussi similaires. Avec ces stratégies, l'on ajoute la notion de *lot* au processus d'orchestration des plans de production.

5.4.1 Regroupement dynamique de lots

Cette stratégie consiste en la création virtuelle et dynamique des lots de produits appartenant à un même ordre de production. L'idée est d'*orienter* tous les produits appartenant à un même ordre de production pour suivre une même route de production et ainsi tenter de réduire le nombre de réglages des ressources et coordonner les efforts d'exploration de HP similaires.

Avec l'arrivée d'un nouvel ordre de production, le gestionnaire des ordres lance une collection des *HP*s comme décrit dans le Chapitre 3. A ce groupe, on assigne un intervalle d'indépendance. Le premier *HP* est désigné comme le leader du lot et se lance dans le processus d'orchestration en créant un graphe de faisabilité et un arbre de solutions faisables. Le reste des *HPs* contribuent aussi à la formation de ce graphe de faisabilité, qui est une donnée globale à l'ordre de production. Cependant, chaque HP explore son propre graphe de solutions faisables (la structure est partagée mais les données d'exploration sont uniques à chaque HP). Lorsqu'un HP explore une solution plus performante par rapport à celles déjà trouvées, alors il la communique au reste des membres du lot. Ceux-ci à leur tour créeront une FdE de profondeur pour l'exploration de cette solution depuis leur perspective. Ce processus continue jusqu'au moment où un HP décide de créer une intention. Le HP communique alors cette décision aux membres du groupe. Ceux ayant la même route comme meilleur candidat dans leur arbre de solutions rejoindront le HP dans sa décision et tenteront de réserver son intention. Ce groupe devient alors un lot identifié par leurs intentions.

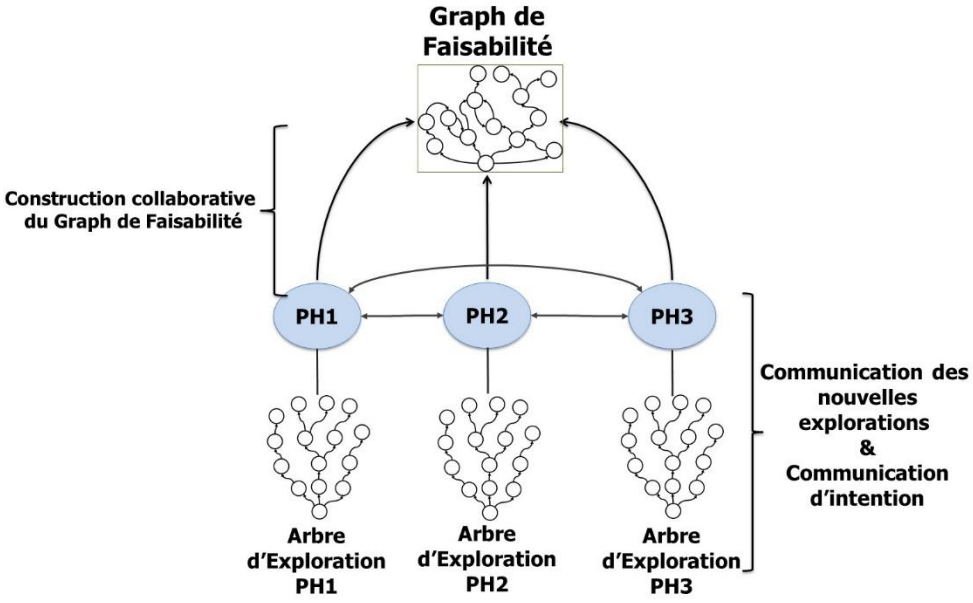


Figure 69 Coopération des Holons Produits

Pendant la phase d'exécution et de reconfiguration, les HP peuvent changer d'intentions en fonction de leur comportement, ce qui divise les lots en plus petits lots, Figure 70 . Ainsi, des lots s'associent et se dissocient pour s'inscrire dans les créneaux d'utilisation tout au long du processus de production.

Ainsi, cette stratégie permet d'« orienter » les plans des HP vers un comportement tendant à ressembler à une production du type flow shop, tous les produits suivant la même trajectoire de production ainsi minimisant les réglages des machines au long de leur parcours, mais sans ignorer les opportunités de parallélisme dans la division des lots. Un autre avantage de cette stratégie est la coopération dans les efforts d'exploration. Tous les HP contribuent à la création du graphe de faisabilité et coopèrent à l'exploration des solutions et trouvent donc plus rapidement des solutions performantes. Cette approche devient spécialement intéressante pour les applications avec centres de décision distribués, dû au potentiel réel de parallélisme dans le calcul.

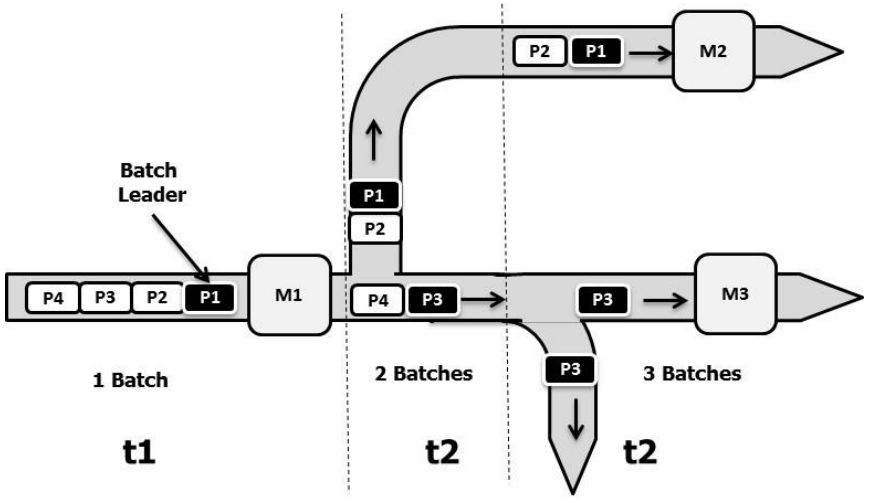


Figure 70 Création dynamique de lots de produits

5.4.2 Requêtes dynamiques de sous-processus

En termes de production, cette stratégie vise à donner priorité à l'exploration de trajectoires minimisant le temps de transports des plans de production. En termes d'efficacité du calcul, cette stratégie vise à réduire considérablement le nombre de sauts dans l'exploration d'une route de production. L'idée est de réduire le nombre les nœuds d'exploration à travers l'encapsulation des branchements d'un processus dans une même requête de service. Autrement dit, cette stratégie cherche à regrouper une séquence des *MServices* pour être exécutés dans une même ressource présentant les capabilités nécessaires pour leur réalisation. Ceci « oriente » la recherche de l'espace de solutions réelles vers des solutions réduisant le temps d'inactivité. Par exemple, pour un ensemble de *n* services indépendants encapsulés dans une même requête de service, le nombre des nœuds à explorer se réduit avec un facteur de (n! - 1).

Pendant la phase d'exploration, lors de l'exploration d'un état, les FdEs observent s'il existe un ensemble de services consécutifs dont l'exécution peut être réalisée par une même ressource. Avant de sélectionner un arc en se basant sur leur fonction heuristique h(a), les fourmis vérifient s'il existe des arcs amenant vers un état dont la ressource correspond à la ressource actuelle (celui en exploration) en se basant sur les données fournies par le DF. Si de tels arcs existent, alors la fourmi en choisit un et se déplace virtuellement dessus pour continuer jusqu'à être obligé de quitter la ressource actuelle pour continuer la production. Alors, la FdE fait une requête de *service composé* qui est la demande non pas d'un service mais d'une séquence de services. En prenant l'exemple de la Figure 71 , dans la requête n° 1, le HP P1 envoie une requête au HR R1 pour l'exécution des services S1 à S5. R1, à son tour, analyse son plan d'utilisation et retourne une réponse proposant uniquement l'exécution de S1 et S2. P1 reçoit cette proposition et explore les étapes subséquentes avec la même démarche, essayant d'attribuer des séquences aux ressources avec capabilités suffisantes. Cette stratégie est spécialement applicable dans des systèmes se caractérisant par des temps de transport relativement longs.

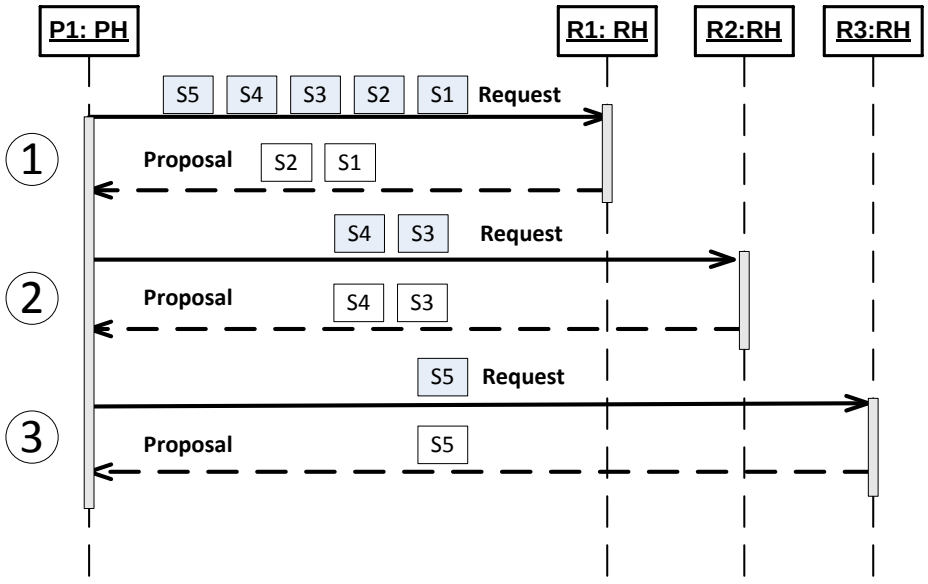


Figure 71 Requêtes de Services Composés et Propositions Alternatives

5.4.3 Discussion des stratégies

Grâce à la contribution de ces deux stratégies (combinées), la performance de l'exploration est augmentée avec l'heuristique de recherche décrite dans la section précédente, étant donné qu'elle prend en compte des informations à jour provenant de l'exploration des FdFs dans un scénario de disponibilité complète. De plus, ces stratégies peuvent être utilisées comme lignes directrices pour réduire la

complexité combinatoire et orienter l'exploration vers la meilleure solution, impliquant moins de changements de séries, moins de transports et des trajectoires les plus linéaires possibles.

Une autre stratégie en perspective de ces travaux est l'utilisation d'ordonnancement opportuniste. Cette approche consiste à identifier, intuitivement, un ensemble d'opérations critiques et trouver un ordonnancement leur convenant. Ensuite, les opérations « non-critiques » essaient d'être intégrées à cet ordonnancement. Si l'une d'entre elles ne peut être insérée, alors elle est incluse dans l'ensemble des opérations critiques et le processus recommence. Cette stratégie a déjà été utilisée dans le contexte holonique dans des travaux comme ceux de (Chové et al., 2009) par exemple.

5.5 CONCLUSION

Ce cinquième chapitre avait pour objectif de présenter notre proposition d'orchestration des plans de production au sien d'un *SoHMS*. Contrairement à d'autres architectures holoniques classiques, nous avons délibérément opté pour une orchestration distribuée, ne passant pas par la création d'holons centralisés dans l'architecture. Une approche inspirée de la *strigmergie* a été proposée, basée sur l'utilisation de colonies de fourmis. De plus, une stratégie permettant de gérer l'explosion combinatoire due à l'utilisation de gammes flexibles au sien du système de pilotage a été proposée.

Après avoir défini la notion d'orchestration, nous avons étudié les approches existantes dans la littérature, ce qui nous a permis de relever un ensemble d'approches répondant partiellement au problème, mais permettant d'aiguiller la conception du système proposé ici. Nous avons ensuite présenté en détail les mécanismes de création puis d'exploration de l'espace de solutions de plans de production à l'aide des clones de fourmis.

Avec cette définition du modèle d'orchestration, le framework général d'implémentation d'un *SoHMS* a été finalisé. Par la suite, nous avons eu l'opportunité de développer une application de ce framework sur un système de production automatisé réel. Cette implémentation sera détaillée dans le chapitre suivant.

Chapitre 6

Mise en Œuvre d'un Cas d'Etudes

Ce dernier chapitre a pour objectif de présenter une application de l'ensemble des concepts et méthodes présentées dans ce document sur une ligne d'assemblage flexible robotisée. Les objectifs de l'implémentation sont multiples. Tout d'abord, une illustration du framework pour à la fois la création d'une ontologie de MServices pour une application spécifique et la conception de gammes de processus basées sur des gammes de produits. De plus, le développement de l'architecture de contrôle sera l'occasion de proposer notre modélisation d'un SoHMS sur un système de production relativement classique. Enfin, ce développement sera également le support pour illustrer les interactions entre holons pour répondre aux besoins de production d'une manière plus tangible, et notamment comment on peut imaginer l'utilisation de modèles flexibles de processus sous forme de réseaux de Petri au sein de ce SoHMS en temps réel.

6.1 CREATION D'UNE ONTOLOGIE D'APPLICATION

Le système présenté (Figure 72) a été choisi pour le développement de ce *SoHMS* en fonction de plusieurs caractéristiques. Tout d'abord, le processus d'assemblage a été choisi pour la possibilité de lancer des expérimentations sans utiliser de consommable. Ensuite, au niveau du produit, l'objectif était de manipuler un assemblage modulaire (permettant donc des décompositions en sous-ensembles), permettant un paramétrage des différents composants, ayant une configuration du produit totalement assemblé qui soit également modulaire et enfin qui soit facilement démontable et réutilisable pour les expérimentations tant en recherche qu'en formation.

Enfin, au niveau de la ligne d'assemblage, l'objectif est d'avoir des ressources polyvalentes de manière à offrir différentes possibilités d'opérations réalisables et ajouter de la flexibilité voire de la redondance au système. Entre chacune de ces ressources, l'idée est de disposer d'un système (automatisé) de transfert multi-routage, permettant une atteignabilité totale depuis tout point du système de transfert. Enfin, il est intéressant de pouvoir disposer de stocks au niveau des ressources afin d'illustrer les modes de gestion de ces éléments au sein du contrôle.

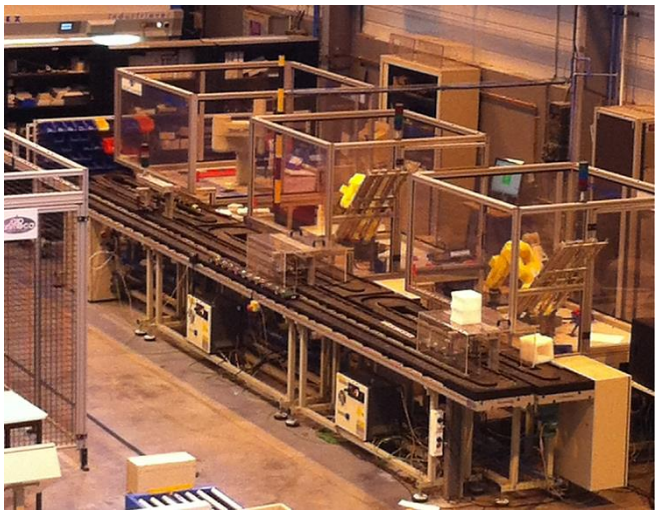


Figure 72 Chaine de production d'implémentation

6.1.1 Les produits

Composition du Produit

Le produit considéré dans cette application (Figure 73) est le même que celui présenté dans le cas d'études de la section 4.5.2. Il représente l'assemblage d'une structure de blocs Lego qui peuvent être empilés dans différentes configurations. L'existence de l'interface d'assemblage universelle Lego permet la mise en œuvre de la modularité du produit final. Dans le domaine physique, les blocs représentent les composants du produit (ou de sous-produits) et les dépendances géométriques entre composants représentent les relations de dépendances généralisables à un assemblage classique. La structure Lego est de plus utilisée comme une référence visuelle des précédences entre composants.

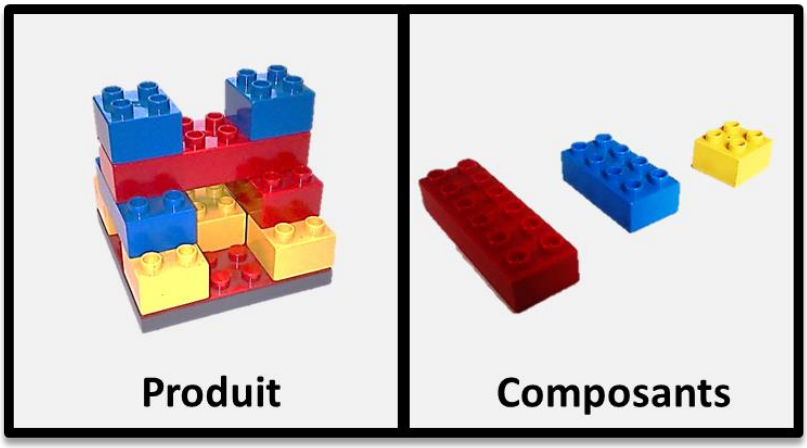


Figure 73 Produit d'étude : structure Lego

Trois types de blocs sont utilisés, de largeur 2 et de longueurs 2, 4 et 6. Chaque bloc est de plus disponible en quatre coloris : rouge, vert, jaune et bleu. Chaque structure est composée d'un maximum de quatre couches de blocs disposés sur une base initiale, de dimension 6x6 et de couleur rouge ou verte. Chaque bloc peut être assemblé dans n'importe quelle position sur la base ou sur d'autres blocs, tant que l'interface inférieure permet une stabilité de l'assemblage.

Paramétrage des produits

Chaque bloc Lego peut être paramétré au sein de la structure en se basant sur 5 paramètres:

1. Coordonnée en X (Figure 74) ;
2. Coordonnée en Y;
3. Coordonnée en Z, i.e. la couche à laquelle le bloc est positionné ;
4. L'orientation W (uniquement pertinent pour les blocs longs et moyens) pouvant valoir 0 ou 90° ;
5. La couleur.

Grâce à ces paramètres, il résulte une large flexibilité pour créer une grande variété de structures et donc de produits.

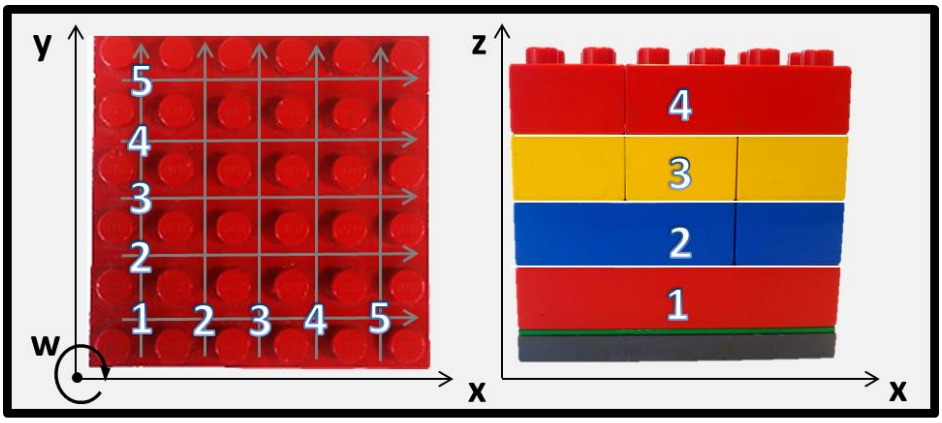


Figure 74 Paramètres de position des blocs

Gammes de produits

Des familles de produits peuvent être construites par la définition de structures de base, sur lesquelles différentes versions peuvent être créées par l'ajout, suppression ou substitution de blocs. Pour des besoins d'illustration, nous définissons ici une famille de produits par une structure de base sur laquelle les positions des blocs sont fixés (i.e., X, Y, Z & W). De ce fait, chaque couleur de bloc peut être modifiée pour créer un nouveau produit appartenant toujours à cette même famille.

De plus, pour la dernière couche du produit, des choix modulaires sont créés pour générer des versions différentes au sein de la famille, permettant ainsi d'inclure des choix d'options. Par exemple, la famille présentée Figure 75 représente une option à choisir entre soit un bloc sur la quatrième couche en position centrale ou de substituer ces blocs par deux blocs positionnés selon la description de la version 2.

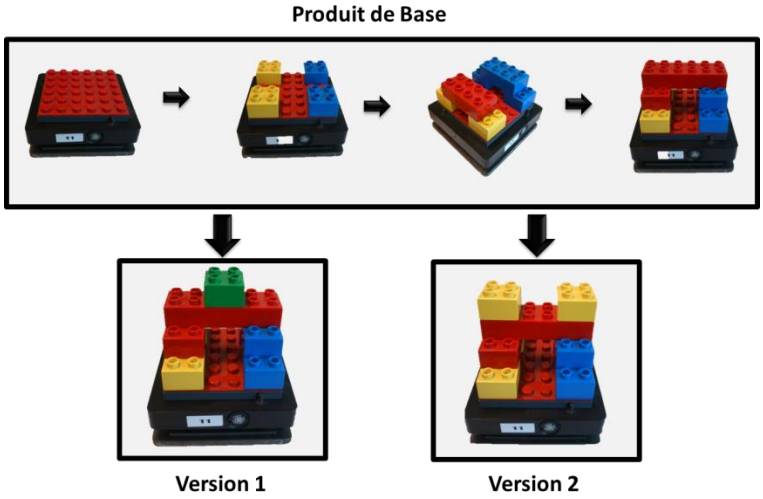


Figure 75 Conception d'une Famille de Produit

Grâce à cela, la personnalisation de cette famille de produits se passe à un niveau scalaire avec le choix de la couleur et à un niveau modulaire avec le choix de la version à fabriquer. Une famille de produit peut alors générer un grand nombre d'instances de produits, identifiés en tant que membres de la famille sur la Figure 76 .

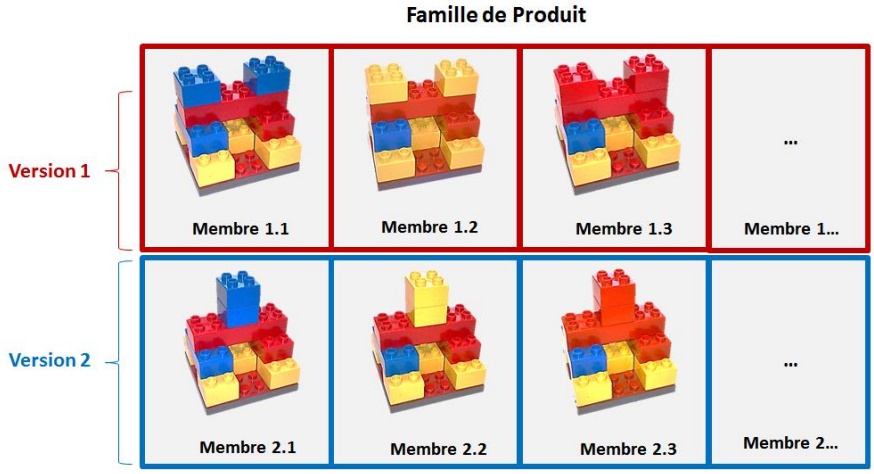


Figure 76 Membres dans une Famille de Produit

6.1.2 Le système de production

Le développement du *SoHMS* a été réalisé sur une ligne d'assemblage située au sein de l'atelier de production du département QLIO de l'IUT de Nantes (Figure 77). Cette ligne a été construite autour d'un système de convoyeurs à segments recyclé de l'atelier de l'Ecole des Mines de Nantes. Quatre convoyeurs ont été disposés, munis de 3 systèmes de transferts entre convoyeurs (échangeurs) et de 3 systèmes de transfert entre les convoyeurs et les postes (feeders). Sur ce convoyeur circulent environ 25 palettes, capables chacune d'accueillir un produit et de le transporter de poste de travail en poste de travail pour compléter sa production. A chaque poste de travail, l'AIP PRIMECA, le laboratoire IRCCyN et l'IUT de Nantes ont installé un robot de type Pick&Place dédié à la manipulation de blocs Lego.

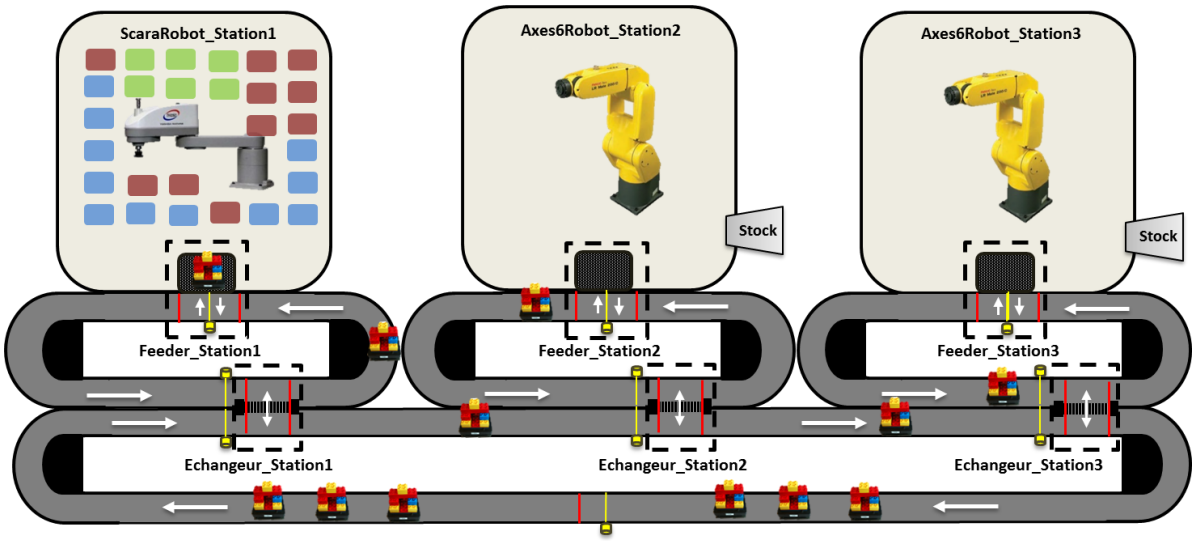


Figure 77 Configuration du système de production pour l'assemblage de blocs Lego

Les postes de travail

Parmi ces postes de travail, deux types peuvent être identifiés. Le poste numéroté 1 (Figure 78) est équipé d'un robot SCARA et d'un stock plan, muni d'emplacement pouvant accueillir des bases, munies ou non de structures de Lego, finies ou semi-finies. Le robot effectue donc des opérations de Pick&Place permettant de déposer des bases sur les palettes vides ou de les retirer.

Les postes 2 et 3 (Figure 79) sont strictement identiques. Ils sont constitués d'un robot FANUC LR Mate 200 équipés de 6 axes et d'une pince électrique. Un stock fixe d'alimentation en blocs Lego (6 emplacements, 2 par taille) et un stock tampon au pied du robot permettent de gérer l'aléa provenant des inadéquations entre couleurs requises et couleurs disponibles en sortie de stock. En effet, les stocks fixes sont constitués de goulottes gravitaires, ce qui fait que seul le bloc en sortie de goulotte est accessible pour la pince du robot. Il faut donc sortir ce bloc et le positionner sur le stock tampon pour accéder au bloc suivant.

Ces robots effectuent également des opérations de type Pick&Place. De manière générale, les robots sont programmés en paramétrés par référence à la position X=1, Y=1, Z=1 des palettes.

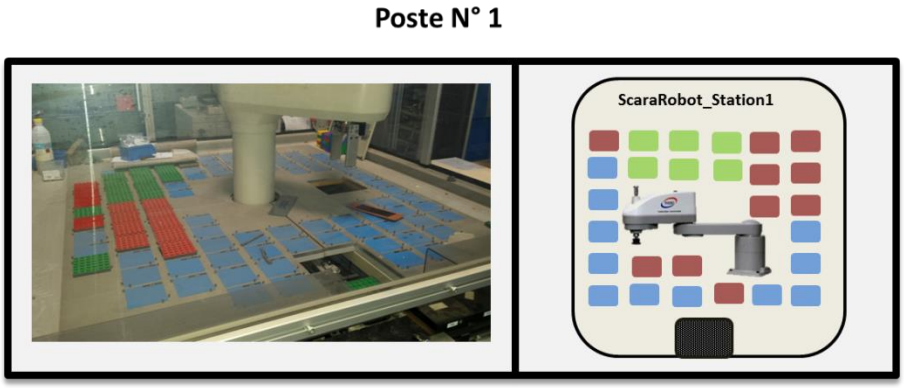


Figure 78 Poste de Travail N°1 : Stockage de produits semi-finis et déchargement de produits finis

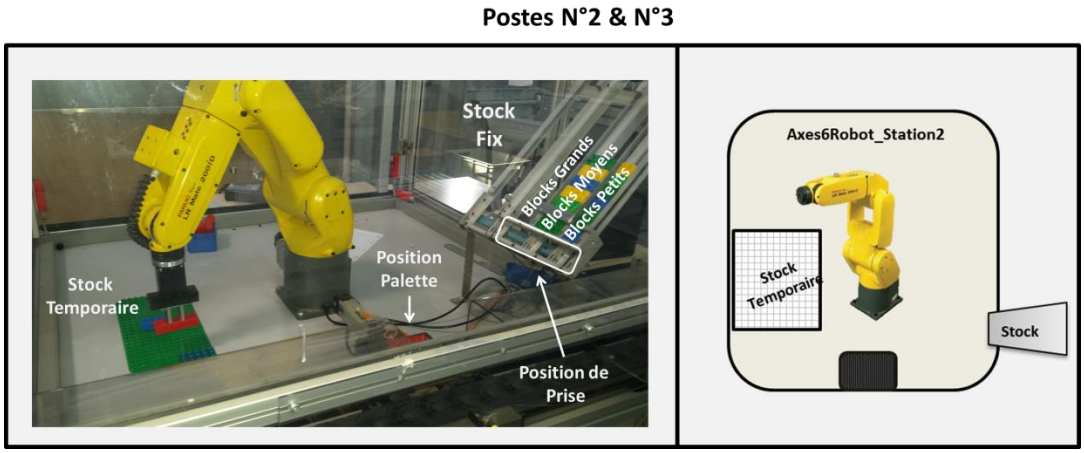


Figure 79 Poste de Travail N°2 et 3 (identiques) : Opérations d'Assemblage de Blocs Lego

Le système de transport

Afin de structurer la présentation du système de transport, les convoyeurs ont été divisés à l'aide de la notion de port (Figure 80). Un port est une zone définie d'un convoyeur, localisée aux positions de prises de décisions et de routage des palettes. Les 4 convoyeurs sont disposés de manière à ce que le plus grand serve de lien entre les 3 plus petits, chacun destiné à desservir un poste de travail. Les palettes sont équipées de capacités d'auto-identification grâce à des étiquettes RFID. Elles sont donc au *niveau 1* de l'intelligence tel que défini par (Wong et al., 2002).

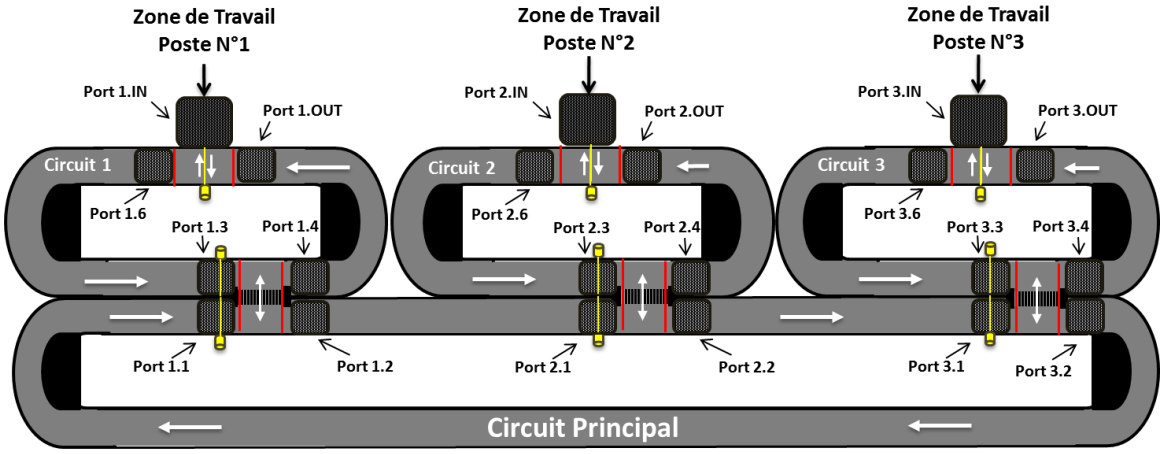


Figure 80 Système de Transport et Localisation des Ports

Les échangeurs (Figure 81) sont des mécanismes automatisés permettant de physiquement transférer une palette d'un convoyeur à l'autre. Une seule palette peut être gérée à la fois. 6 capteurs de présence sont nécessaires pour l'automatisation et le séquencement des palettes grâce aux bloqueurs et deux lecteurs RFID sont installés pour l'identification des palettes aux ports #.1 et #.3. Enfin, un vérin double effet permet de tirer ou pousser les palettes perpendiculairement à l'axe des convoyeurs.

Les feeders (Figure 82) sont construits autour d'un vérin double effet permettant de tirer les palettes à l'intérieur des postes, deux capteurs, deux bloqueurs et un lecteur RFID.

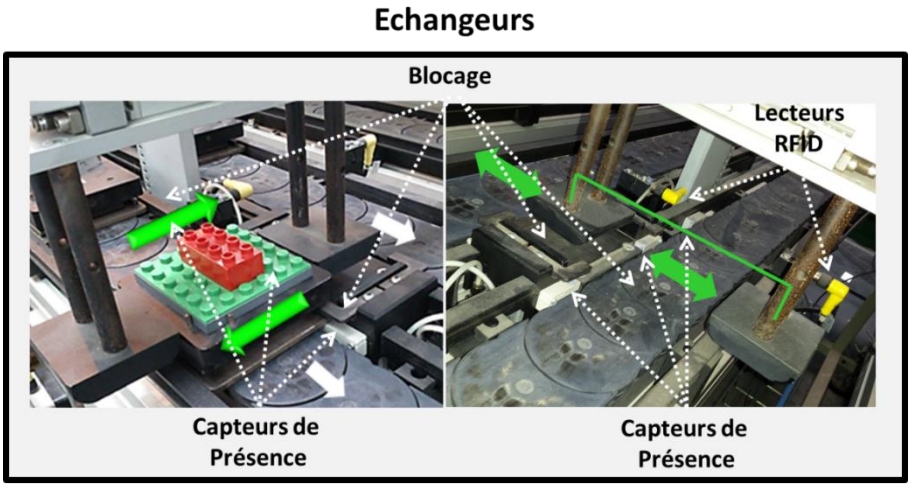


Figure 81 Mécanisme de Transfert inter-convoyeurs

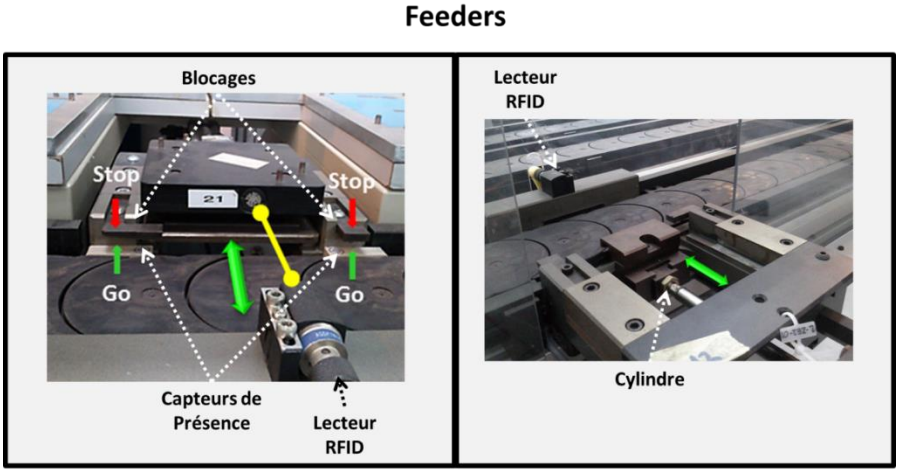


Figure 82 Mécanisme de transfert des palettes aux postes de travail.

6.1.3 L'architecture du système

Le système présenté dans la section précédente sera définitivement prêt en Novembre 2015, soit après la rédaction de ce document. Afin de valider le comportement du *SoHMS*, une phase de « *virtual commissioning* » a été nécessaire (Berger et al., 2015).

Conception d'un système d'émulation

Nous avons donc conçu un émulateur du système en simulation à évènements discrets. Ce modèle d'émulation a été implémenté sur Rockwell Arena, l'interface de communication étant gérée par la couche Visual Basic pour Applications à l'aide de TCP sockets. De tels outils sont tout à fait appropriés pour modéliser des activités basées sur la gestion des files d'attente. Dans ce modèle, les files d'attente sont énormément utilisées, à la fois pour la modélisation du système de convoyage et pour la synchronisation entre les évènements se déroulant sur l'émulation et les ordres provenant du *SoHMS* (Figure 83).

Le modèle ne contient aucune intelligence, et n'est capable que d'exécuter une liste préprogrammée d'ordres en réaction à une demande sémantiquement de haut niveau sur le socket d'interface. Il est capable de comprendre des ordres tels que "TRANSPORT TransporterID FROM InitialZone TO FinalZone" ou "PICK RobotID Store" (Table 4). Les actions des robots sont émulées par des délais de durée prédéterminée.

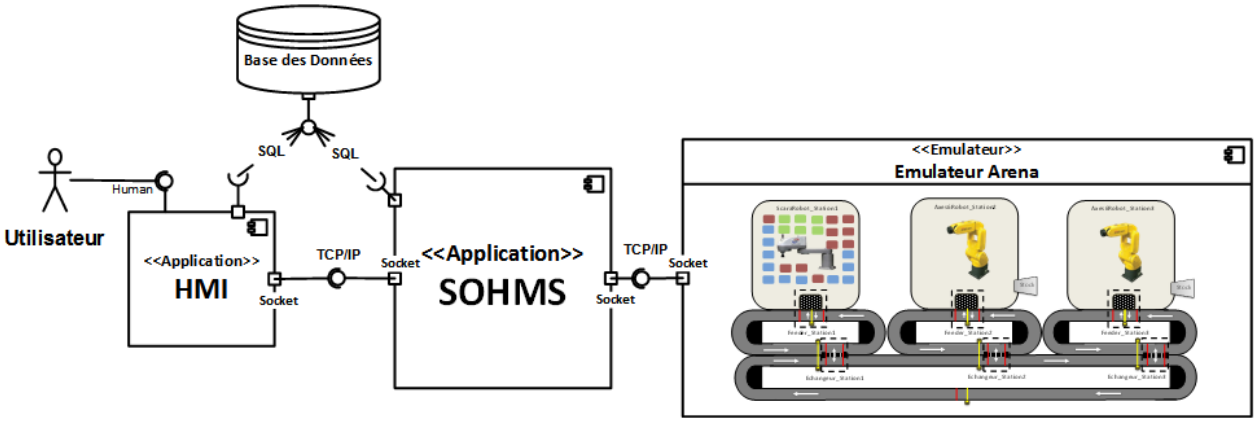


Figure 83 Architecture d'émulation

Table 4 List des Commandes envoyés à MLM

Commandes SoHMS→ MLM

Action	Structure du Message	Exemple
Ordonne à une ressource de transporter la palette du port initial vers un port final.	« TRANSPORT» RessourceID PortInitial PortFinal	TRANSPORT 1 Port1.1 Port1.2
Opération du Robot pour prendre un bloc du stock à la position indiquée par StockID.	« PICK » PosteID StockID	PICK Poste2 Rack1
Ordonne au robot de positionner le bloc à la position indiquée par X Y Z W.	« PLACE » PosteID X Y Z W	PLACE Poste2 3 3 1 0

Table 5 List des Notifications remontés au *SIL* du *SoHMS*

Notifications MLM→ SIL

Notifications	Structure du Message	Exemple
Indique la détection d'une palette à l'entrée ou sortie de la ressource indiquée par RessourceID.	PALLET ID RessourceID Port	PALLET 10 Ex1 Port1.1
Notification de réception de la commande.	« OK » Commande	OK TRANSPORT 1 Port1.1 Port1.2
Notification de Début d'exécution de la commande.	« START » Commande	START TRANSPORT 1 Port1.1 Port1.2
Notification de terminaison de l'exécution de la commande.	« END » Commande	END TRANSPORT 1 Port1.1 Port1.2

Conception du système réel

Le choix fait sur ce système a été de remplacer les classiques automates programmables industriels (API) par des modules d'entrées-sorties sur IP Beckhoff. Ces modules ne sont que des passerelles multi-protocoles (gérant notamment le RS232 pour la lecture RFID, MODBUS TCP pour la partie IP…) et ne contiennent aucun programme de contrôle tel que ceux que l'on peut rencontrer dans les API classiques. Ceux-ci sont donc remplacés par des programmes ad hoc, capables de gérer des sémantiques de plus haut niveau que ce que les automates permettent et plus flexibles dans la partie configuration pour la partie expérimentale (Figure 84).

Tout d'abord, un middleware de bas niveau (Low-Level Middleware – LLM) a été créé. L'objectif de LLM est de récupérer de manière synchrone l'état de l'ensemble des capteurs du système, informer de manière asynchrone les couches hautes du contrôle de tout changement de valeurs de ces capteurs et modifier de manière asynchrone l'état des actionneurs du système sur ordre des couches hautes. Fonctionnellement, ces fonctionnalités sont proches de ce qu'un serveur OPC[8] pourrait faire, mais adapté à la configuration matérielle.

Puis, un middleware de niveau intermédiaire (Medium-Level Middleware – MLM) a été créé pour prendre en charge l'agrégation de données provenant de LLM en direction des couches hautes du contrôle et temporiser les macro-actions demandées par les couches haute dans une sémantique de haut niveau en actions élémentaires exécutées ensuite par LLM, comme illustré par la table X. Par exemple, quand un service de Pick est demandé à un robot, cette requête de service est transmise à MLM, qui communique à LLM tous les octets de configuration à modifier sur le contrôleur du robot, puis attend les confirmations de modification déclenchées par la surveillance des données de LLM. MLM peut ensuite déclencher le démarrage du programme sur le contrôleur robot, mais doit pour cela suivre un protocole précis de mises à 1 et à 0 de données dans un timing constitués d'intervalles de temps inférieurs à 100ms. Lorsque le programme est lancé, MLM envoie un acquittement de démarrage du service à la couche supérieure, attend l'information provenant de LLM comme quoi le programme est terminé puis envoie lui-même l'information de fin de service. Ces fonctionnalités sont proches cette fois-ci de celles exécutées par un API, mais encore une fois elles permettent une complète flexibilité et une utilisation de sémantique de plus haut niveau.

[8] www.opcfoundation.org

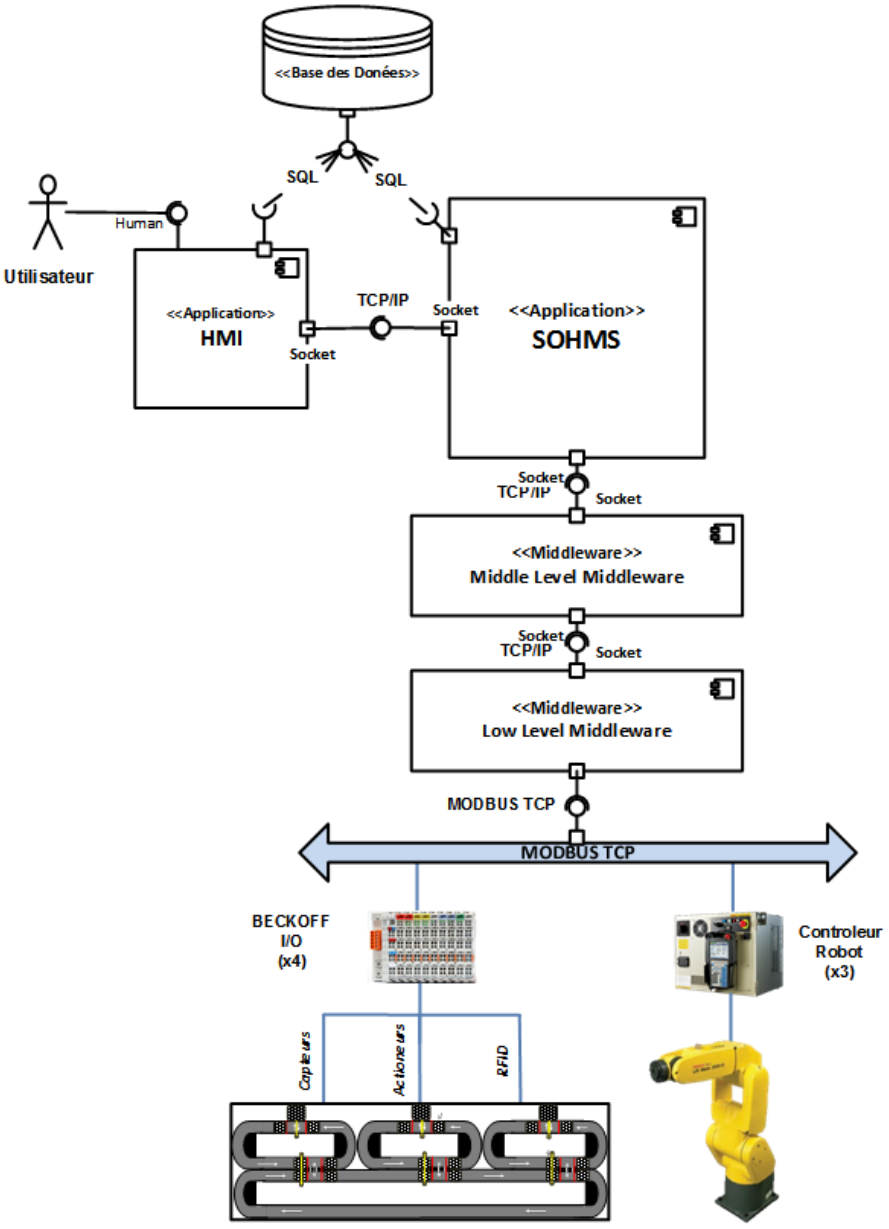


Figure 84 Architecture du système réel

Conception d'une Interface Homme-Machine

En parallèle de ces travaux, nous avons développé une interface homme-machine (HMI) basée sur les principes de supervision du système. Son objectif est de regrouper toutes les fonctions de paramétrage nécessaires au bon fonctionnement de *SoHMS* (Figure 85), ainsi que des outils de suivi de la production. Cette interface a l'objectif de servir à la fois en recherche et en pédagogie (Bac+1 à +3), d'où certaines adaptations afin de simplifier son utilisation. Pour le développement de cet outil, nous avons reçu le soutien de deux stagiaires de niveau Bac+2 DUT Informatique de l'IUT de Nantes (B. Robert et A. Bouchet).

Les fonctions générales demandées à HMI sont :

- La conception de produits adaptés à l'application Lego (Figure 86) ;
- L'édition des gammes de production flexibles (encore en cours de développement)
- La définition des ordres de production (Figure 87) ;

- Association d'un produit précédemment conçu à l'ordre ;
- Définition des paramètres de l'ordre : échéance, nombre de produits à fabriquer, nombres de palettes utilisées ;
- Transmission de l'ordre à un *SoHMS* précédemment connecté.

Figure 85 Menu de configuration – HMI

Figure 86 Conception de Produits - HMI

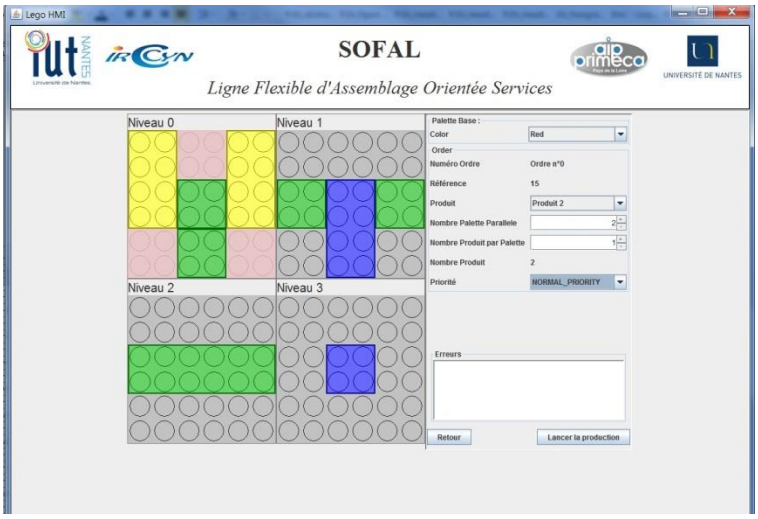


Figure 87 Création et Lancement des Ordres de Production – HMI

6.2 PRESENTATION DU MODELE *SOHMS*

6.2.1 Les services de production

Cette section a pour objectif d'illustrer la modélisation des services de production dans l'architecture *SoHMS*, et notamment les ontologies d'application créées pour l'implémentation étudiée.

Représentation des MServices

Dans une section précédente, nous avons présenté pour illustration une première version des idées présentées dans la version définitive de l'application. Chaque bloc Lego dans la structure de la famille de produits représente ainsi une instance de *MService*. Il y a donc une identification 1 pour 1 entre les blocs et les services de production (Figure 88).

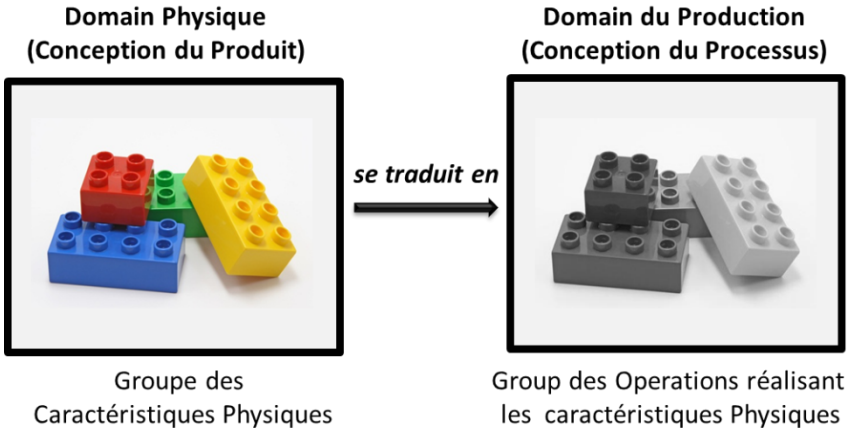


Figure 88 Exemples de blocs Lego représentant des *MServices* avec leurs relations d'interdépendance

De manière à générer une instance d'application plus large et intéressante, quelques modifications sont apportées par rapport à la présentation précédente du problème. Ainsi, les types des services sont caractérisés par le couple {type de bloc ; couche}, ce qui donne une ontologie de 4x3 = 12 types de services : A1, B1, C1, A2, B2, C2, A3, B3, C3, A4, B4, C4 (Figure 89). Les paramètres restent par contre les mêmes: X, Y, Z, W et la couleur.

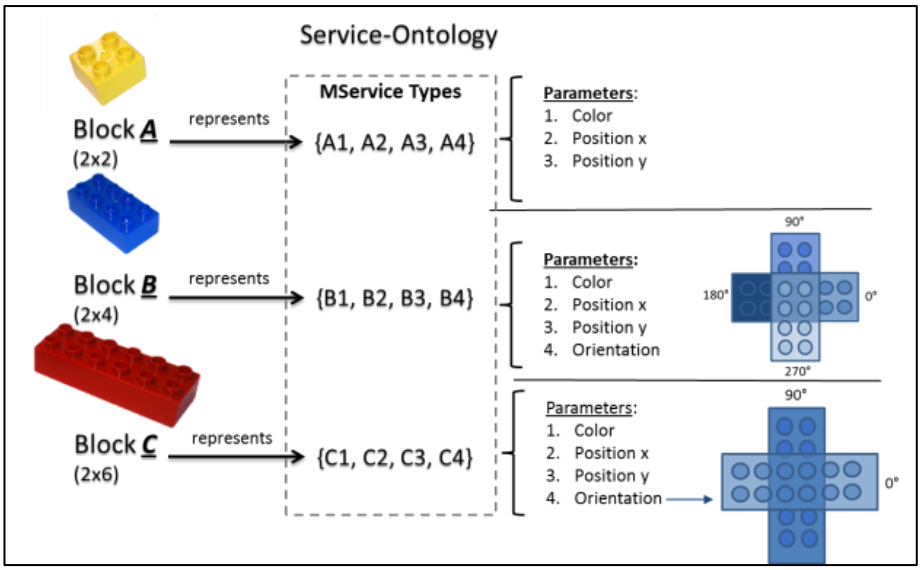


Figure 89 Ontologie Lego des *MServices* (Rappel de la figure dans la section 4.5.2)

Autres Types de Services

D'autres types de service ont également été créés, notamment celui permettant de charger/décharger une palette en début/fin de production ou celui permettant de déplacer les palettes au sein du système de transfert :

« Supply Base » : ce service représente l'opération d'ajouter une base munie de l'interface Lego sur laquelle les blocs seront assemblés. Le seul paramètre de ce service est la couleur (rouge ou vert).

« Transport Service » : Il représente l'opération de faire bouger une palette (éventuellement transportant un produit) d'un port à l'autre du système de transfert. Chaque port est associé à une ressource active du système. Ce service n'appartient pas à l'ontologie Lego mais plutôt à l'ontologie « *TransportServiceOntology* ». Les paramètres associés à ce service sont :

- *StartPort :* port initial ;
- *FinalPort :* port de destination.

Déclaration XML des ontologies

Deux ontologies de service ont donc été créées et sont présentées dans les figures suivantes sous la forme d'une déclaration XML : une ontologie axée sur l'assemblage des blocs Lego (Figure 91) et une ontologie axée sur le transfert des palettes (Figure 90). Ces ontologies, comme indiqué précédemment, permettent de structurer la déclaration des services. Ainsi, les Figure 92 et Figure 93 présentent des exemples de déclaration de services conformes à chacune des ontologies suscitées.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ServiceOntology Name= "TransportServiceOntology">
    <Services>
        <MServiceType Name="TransportPallet" Ontology="Transport"
                      Category="Transport" Taxonomy="none"
                      ContactInfo="francisco.gamboa@univ_nantes.fr">
            <Description>Takes a pallet from an Initial Port to a
                         Final Port in the System's Layout Map</Description>
            <Parameters>
                <Parameter DataType="String">
                    <Name>StartPort</Name>
                </Parameter>
                <Parameter DataType="String">
                    <Name>FinalPort</Name>
                </Parameter>
            </Parameters>
            <Attributes>
                <Attribute DataType="Double">
                    <Name>Cmax</Name>
                </Attribute>
                <Attribute DataType="String">
                    <Name>Quality</Name>
                </Attribute>
            </Attributes>
        </MServiceType>
   </Services>
</ServiceOntology>
```

Figure 90 Ontologie des services de transport

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ServiceOntology Name= "Lego ServiceOntology">
    <Services>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceType Name="ServiceA" Ontology="LegoBlocks"
                      Category="Assembly" Taxonomy="none"
                      ContactInfo="francisco.gamboa@univ-nantes.fr">
            <Description>Assembles a block of size 2x2</Description>
            <Parameters>
                <Parameter DataType="String">
                    <Name>color</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>X</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>Y</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>Z</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>W</Name>
                </Parameter>
            </Parameters>
            <Attributes>
                <Attribute DataType="Double">
                    <Name>Cmax</Name>
                </Attribute>
                <Attribute DataType="String">
                    <Name>Quality</Name>
                </Attribute>
            </Attributes>
        </MServiceType>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceType Name="DispatchProduct" Ontology="LegoBlocks"
                      Category="Dispatch" Taxonomy="none"
                      ContactInfo="francisco.gamboa@univ-nantes.fr">
            <Description>Dispatch the Finish Product to a Stock of
                      Finished Products</Description>
            <Parameters/>
            <Attributes>
                <Attribute DataType="Double">
                    <Name>Cmax</Name>
                </Attribute>
                <Attribute DataType="String">
                    <Name>Quality</Name>
                </Attribute>
            </Attributes>
        </MServiceType>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceType Name="SupplyBase" Ontology="LegoBlocks"
                      Category="Supply" Taxonomy="none"
                      ContactInfo="francisco.gamboa@univ-nantes.fr">
            <Description>Supply Lego Base of a certain color
                      to Pallet</Description>
            <Parameters>
                <Parameter DataType="String">
                    <Name>color</Name>
                </Parameter>
            </Parameters>
            <Attributes>
                <Attribute DataType="Double">
                    <Name>Cmax</Name>
                </Attribute>
                <Attribute DataType="String">
                    <Name>Quality</Name>
                </Attribute>
            </Attributes>
        </MServiceType>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceType Name="ServiceB" Ontology="LegoBlocks"
                      Category="Assembly" Taxonomy="none"
                      ContactInfo="francisco.gamboa@univ-nantes.fr">
            <Description>Assembles a block of size 2x4</Description>
            <Parameters>
                <Parameter DataType="String">
                    <Name>color</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>X</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>Y</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>Z</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>W</Name>
                </Parameter>
            </Parameters>
            <Attributes>
                <Attribute DataType="Double">
                    <Name>Cmax</Name>
                </Attribute>
                <Attribute DataType="String">
                    <Name>Quality</Name>
                </Attribute>
            </Attributes>
        </MServiceType>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceType Name="ServiceC" Ontology="LegoBlocks"
                      Category="Assembly" Taxonomy="none"
                      ContactInfo="francisco.gamboa@univ-nantes.fr">
            <Description>Assembles a block of size 2x6</Description>
            <Parameters>
                <Parameter DataType="String">
                    <Name>color</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>X</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>Y</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>Z</Name>
                </Parameter>
                <Parameter DataType="Integer">
                    <Name>W</Name>
                </Parameter>
            </Parameters>
            <Attributes>
                <Attribute DataType="Double">
                    <Name>Cmax</Name>
                </Attribute>
                <Attribute DataType="String">
                    <Name>Quality</Name>
                </Attribute>
            </Attributes>
        </MServiceType>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
    </Services>
</ServiceOntology>
```

Figure 91 Ontologie des services d'assemblage de Lego

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourceHolon Name="Poste2" Technology="Robot 6 Axis" Category="Assembler">
    <Description>Robotic arm with 6 axis and equipped of a pneumatic clamp</Description>
    <InputPorts>
        <Port>Port2.IN</Port>
    </InputPorts>
    <OutputPorts>
        <Port>Port2.IN</Port>
    </OutputPorts>
    <Service_Offer>
    <!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="ServiceA" Ontology="LegoBlocks"
                          Category="Assembly" Taxonomy="none"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
                <Description>Assembles a block of size 2x2</Description>
            </MServiceType>
            <ParameterProfileSet>
                <Parameters>
                    <ParameterProfile RangeType="Catalogue" DataType="String">
                        <Name>Color</Name>
                        <RangeValues>
                            <Unit value="Green"/>
                            <Unit value="Blue"/>
                            <Unit value="Red"/>
                        </RangeValues>
                    </ParameterProfile>
                    <ParameterProfile RangeType="Interval" DataType="Integer">
                        <Name>X</Name>
                        <RangeValues>
                            <Interval min="0" max="4"/>
                        </RangeValues>
                    </ParameterProfile>
                    <ParameterProfile RangeType="Interval" DataType="Integer">
                        <Name>Y</Name>
                        <RangeValues>
                            <Interval min="0" max="4"/>
                        </RangeValues>
                    </ParameterProfile>
                    <ParameterProfile RangeType="Interval" DataType="Integer">
                        <Name>Z</Name>
                        <RangeValues>
                            <Interval min="0" max="3"/>
                        </RangeValues>
                    </ParameterProfile>
                </Parameters>
                <Methods>
                    <MethodID>21</MethodID>
                    <MethodID>22</MethodID>
                </Methods>
            </ParameterProfileSet>
            <Method ProcessType="Atomic" ID="21">
                <SetupID>1</SetupID>
            </Method>
            <Method ProcessType="Atomic" ID="22">
                <SetupID>2</SetupID>
            </Method>
            <AverageCost>20</AverageCost>
        </MServiceImplementation>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="ServiceB" Ontology="LegoBlocks"
                          Category="Assembly" Taxonoy="none"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
                <Description>Assembles a block of size 2x4</Description>
            </MServiceType>
            .
            .
            .
        </MServiceImplementation>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="ServiceC" Ontology="LegoBlocks"
                          Category="Assembly" Taxonomy="none"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
                <Description>Assembles a block of size 2x6</Description>
            </MServiceType>
            .
            .
            .
        </MServiceImplementation>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="SupplyBase" Ontology="LegoBlocks"
                          Category="Supply" Taxonomy="sometax"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
            <Description>Supply Lego Base of a certain color to Pallet</Description>
            .
            .
            .
        </MServiceImplementation>
        <!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
    </Service_Offer>
</ResourceHolon>
```

Figure 92 Exemple de déclaration de service A pour le poste 2

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RouterRH Name="Exchanger1" Technology="Exchanger" Category="Transport">
    <Description>An Exchanger between Conveyors</Description>
    <InputPorts>
        <Port>Port1.1</Port>
        <Port>Port1.3</Port>
    </InputPorts>
    <OutputPorts>
        <Port>Port1.2</Port>
        <Port>Port1.4</Port>
    </OutputPorts>
    <Service_Offer>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="TransportPallet" Ontology="Transport"
                          Category="Transport" Taxonomy="none"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
            <Description>Takes a pallet from an Initial Port to a
            Final Port in the System's Layout Map</Description>
        </MServiceType>
            <ParameterProfileSet>
                <Parameters>
                    <ParameterProfile RangeType="Catalogue" DataType="String">
                        <Name>StartPort</Name>
                        <RangeValues>
                            <Unit value="Port1.1"/>
                        </RangeValues>
                    </ParameterProfile>
                     <ParameterProfile RangeType="Catalogue" DataType="String">
                        <Name>FinalPort</Name>
                        <RangeValues>
                            <Unit value="Port1.2"/>
                        </RangeValues>
                    </ParameterProfile>
                </Parameters>
                <Methods>
                    <MethodID>1</MethodID>
                </Methods>
            </ParameterProfileSet>
            <!--::..............................:-->
             <ParameterProfileSet>
               ...
                       <Name>StartPort</Name>
                       ...
                           <Unit value="Port1.1"/>
                       ...
                       <Name>FinalPort</Name>
                       ...
                           <Unit value="Port1.4"/>
                       ...
            </ParameterProfileSet>
             <!--::..............................:-->
             <!--::..............................:-->
             <ParameterProfileSet>
               ...
                       <Name>StartPort</Name>
                       ...
                           <Unit value="Port1.3"/>
                       ...
                       <Name>FinalPort</Name>
                       ...
                           <Unit value="Port1.2"/>
                       ...
            </ParameterProfileSet>
             <!--::..............................:-->
             <ParameterProfileSet>
               ...
                       <Name>StartPort</Name>
                       ...
                           <Unit value="Port1.3"/>
                       ...
                       <Name>FinalPort</Name>
                       ...
                           <Unit value="Port1.4"/>
                       ...
            </ParameterProfileSet>
             <!--::..............................:-->
            <Method ProcessType="Atomic" ID="1">
                <SetupID>1</SetupID>
            </Method>
            <Method ProcessType="Atomic" ID="2">
                <SetupID>1</SetupID>
            </Method>
            <Method ProcessType="Atomic" ID="3">
                <SetupID>2</SetupID>
            </Method>
            <Method ProcessType="Atomic" ID="4">
                <SetupID>2</SetupID>
            </Method>
        </MServiceImplementation>
      <!--:::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
    </Service_Offer>
</RouterRH>
```

Figure 93 Exemple de déclaration de Service de Transport Pour l'Echangeur 1

Génération de gammes basées sur les services de production

La fonctionnalité permettant de générer automatiquement les gammes flexibles basées sur les RdP n'a pas encore été développée et insérée dans le démonstrateur. Néanmoins, les chapitres précédents ont présenté tous les éléments nécessaires à générer automatiquement ces gammes à partir de la table de précédences entre services (Figure 94). Toutefois, le système de commande est déjà capable de parcourir des gammes créées sans utilisation de HMI.

Manufacturing Service Type	Uphill Dependancies	Downhill Dependencies
Service-type 1(Sr 1,1)	-	Sr3
Service-type 1(Sr 1,2)	-	Sr3
Service-type 2(Sr 2,1)	-	Sr4
Service-type 2(Sr 2,2)	-	Sr4
Service-type 3(Sr 3)	(Sr1,1 & Sr1,2)	Sr5
Service-type 4(Sr 4)	(Sr2,1 & Sr2,2)	Sr5
Service-type 5(Sr 5)	(Sr3 & Sr4)	(Sr6) | (Sr7,1 & Sr7,2)
Service-type 6(Sr 6)	Sr5	-
Service-type 7(Sr 7,1)	Sr5	-
Service-type 7(Sr 7,2)	Sr5	-

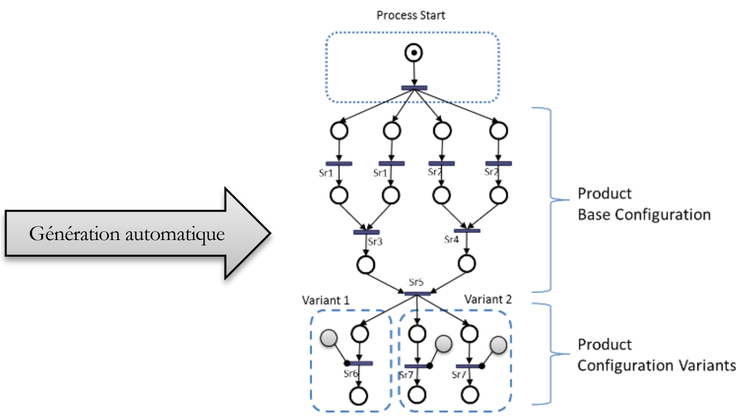
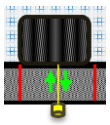
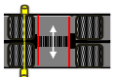
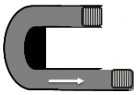
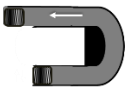


Figure 94 Exemple de table de dépendances entre *MServices* et sa correspondance en RdP

6.2.2 Modélisation holonique

Cette section a pour objectif de décrire synthétiquement l'architecture holonique mise en place. Afin de structurer cette description, la Figure 95 propose une mise en parallèle des holons Produit et des holons Ressources spécialisés. La Figure 96 propose un diagramme de classe complet illustrant les relations existantes entre ces holons.

<table>
<tr><th colspan="3">Modélisation SoHMS : Application Lego</th></tr>
<tr><td>Holon Produit</td><td>Description</td><td>Instances</td></tr>
<tr><td>  </td><td>• Produit individuel appartenant à un ordre de production
• Le Produit est composé d'une partie virtuelle, une partie physique (le produit Lego lui-même) et son association à un transporteur (palette) qui l'accompagnera pendant tout son cycle de vie.</td><td>Produits lancés par des ordres de production en exécution.</td></tr>
<tr><td colspan="3">Holons Ressource</td></tr>
<tr><td>Router HR</td><td>Description</td><td>Instances</td></tr>
<tr><td>  </td><td rowspan="2">• Mécanismes dont la fonction est de transférer une palette d'un port d'entrée vers un port de sortie de la même ressource.
• Il peut y avoir plus d'un port d'entrée et plus d'un port de sortie par ressource
• Ne peut traiter qu'un seul produit à la fois
• L'action de transfert nécessite une position initiale connue
• S'il y a présence de plusieurs palettes à la fois, le RouterRH donne préférence aux transferts partant de la position initiale actuelle du mécanisme de routage.
• Lorsqu'une requête de service est reçue, le routeur demande un permis de port aux ressources propriétaires du port de destination. L'exécution du service se fait lorsque tous les propriétaires accordent le permis de port.
• Dans le cas où cela échoue, (au bout de plusieurs tentatives de la part du Holon Produit) le RouterRH force une action par défaut pour éviter des blocages.
• Comportement purement réactif sans planning de réservation.</td><td>Fed1
Fed2
Fed3</td></tr>
<tr><td>  </td><td>Ex1
Ex2
Ex3</td></tr>
<tr><td>Buffered HR</td><td>Description</td><td>Instances</td></tr>
</table>

	• Segment de convoyeur unidirectionnel. • Le convoyeur transporte des palettes d'un port d'entrée à un port de sortie. • Capacité cumulative d'un nombre de palettes proportionnel à sa longueur. • Lorsque nécessaire, elle accorde des permis de port aux RouterRH qui se trouvent à l'entrée. • Ressource passive, sans capacité de commande. • Lorsqu'un produit se présente à son port d'entrée, la requête de service est exécutée automatiquement. • Comportement purement réactif sans planning de réservation.	conveyor1.1 conveyor2.1 conveyor3.1
		conveyor1.2 conveyor2.2 conveyor3.2
		Conveyor0.1 Conveyor0.2
		Conveyor0.3
Simple HR	Description	Instances


 
	• Postes de travail offrant des opérations de transformation du produit.
	• Capacité d'opérer sur un seul produit à la fois.
	• Possède un comportement capable de répondre aux requêtes de service de manière réactive et/ou prédictive.
	  ○ Mode Prédictif : La ressource inscrit des réservations dans son planning d'utilisation.
	  ○ Mode Réactif : La ressource répond aux requêtes de service *ssi* elle ne met pas en conflit le plan d'utilisation prévu. En cas de conflit, elle force une action par défaut pour libérer le port d'entrée et pouvoir continuer en accord aux plannings.
	•
		Post1
		Post2
		Post3

Figure 95 Correspondance entre holons et représentation physique

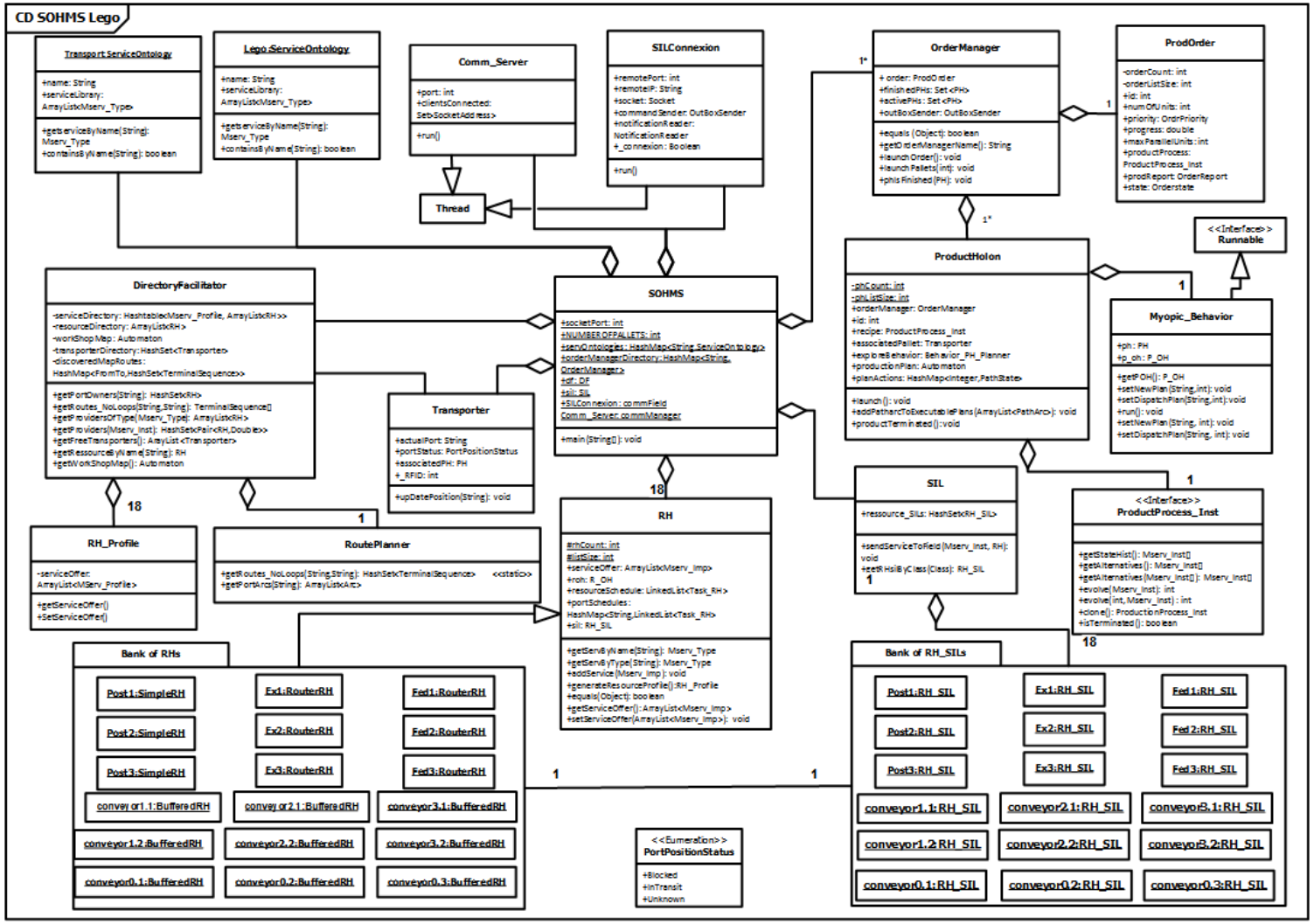


Figure 96 Architecture du *SoHMS* pour l'application Lego

6.2.3 Configuration du système

La configuration du système Figure 97 a pour objectif de se faire de manière simple avec une réduction significative des efforts de programmation pour l'intégration des nouvelles ressources. Dans le système de pilotage, l'intégration se fait par la déclaration des capacités des ressources. Ceci se fait à travers des fichiers XML contenant le « Ressource Profile ». Avec cette information, le système de pilotage peut gérer l'utilisation de la ressource et la rendre disponible pour les autres acteurs.

Tout ce qui est programmable est concentré dans un seul module : le « *Service Interface Layer* » (*SIL*). Le *SIL* est le seul composant de programmation qui nécessite des efforts de développement directement relatifs à l'application. Du à sa spécificité, un *SIL* doit être créé pour chaque nouveau type de ressource afin de faire le lien entre services d'une ontologie et les commandes du contrôleur de la ressource au niveau terrain, propres à l'atelier. Ces efforts sont réduits avec la création d'une bibliothèque de *SIL* selon le type de ressource et une ontologie de services.

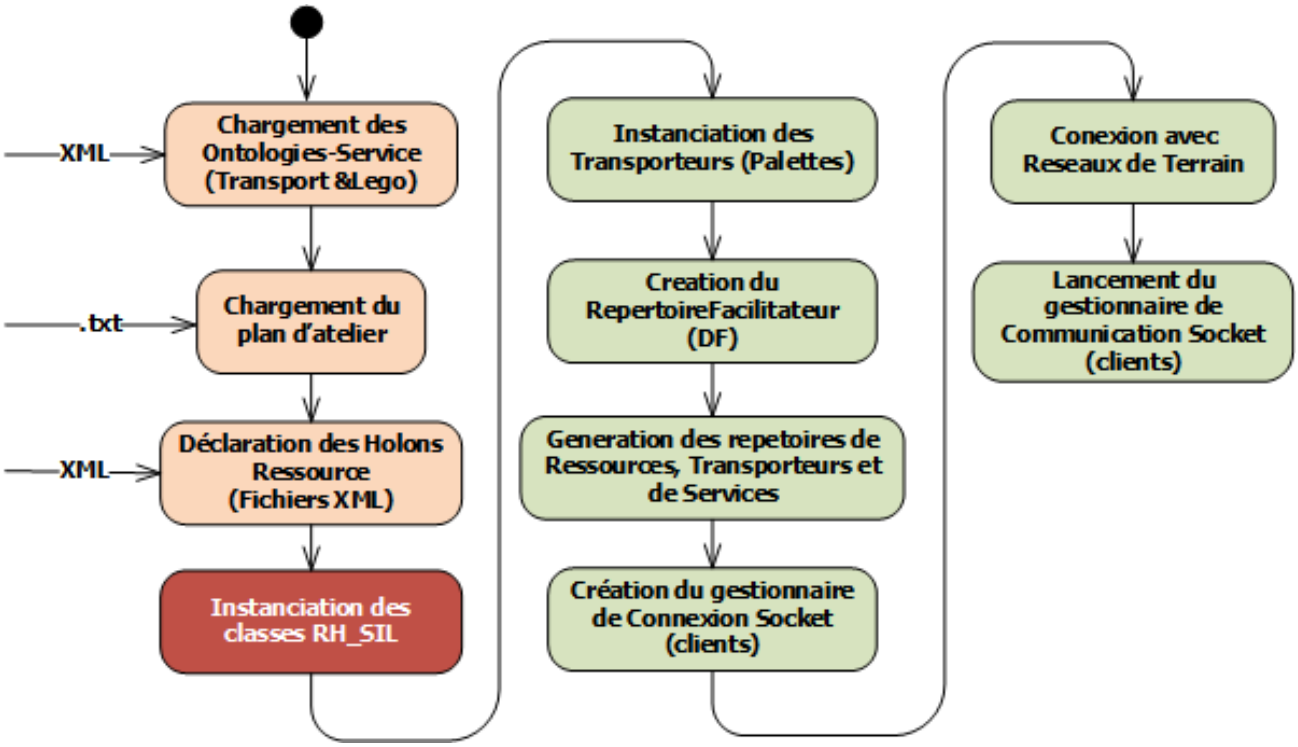


Figure 97 Initialisation du *SoHMS*

Pour le moment, la configuration est réalisée au lancement du *SoHMS* en instanciant des fichiers XML contenant la déclaration des Ressources et le plan de l'atelier. Cette action sera à terme migrée vers l'HMI où l'on pourra à travers l'interface graphique ajouter et retirer des Ressources du Système.

Déclaration des postes

La Figure 98 présente pour illustration le fichier XML généré et transmis au *SoHMS* pour la déclaration d'un poste de travail et de ses capacités.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourceHolon Name="Poste2" Technology="Robot 6 Axis" Category="Assembler">
    <Description>Robotic arm with 6 axis and equipped of a pneumatic clamp</Description>
    <InputPorts>
        <Port>Port2.IN</Port>
    </InputPorts>
    <OutputPorts>
        <Port>Port2.IN</Port>
    </OutputPorts>
    <Service_Offer>
    <!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="ServiceA" Ontology="LegoBlocks"
                          Category="Assembly" Taxonomy="none"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
                <Description>Assembles a block of size 2x2</Description>
            </MServiceType>
            <ParameterProfileSet>
                <Parameters>
                    <ParameterProfile RangeType="Catalogue" DataType="String">
                        <Name>Color</Name>
                        <RangeValues>
                            <Unit value="Green"/>
                            <Unit value="Blue"/>
                            <Unit value="Red"/>
                        </RangeValues>
                    </ParameterProfile>
                    <ParameterProfile RangeType="Interval" DataType="Integer">
                        <Name>X</Name>
                        <RangeValues>
                            <Interval min="0" max="4"/>
                        </RangeValues>
                    </ParameterProfile>
                    <ParameterProfile RangeType="Interval" DataType="Integer">
                        <Name>Y</Name>
                        <RangeValues>
                            <Interval min="0" max="4"/>
                        </RangeValues>
                    </ParameterProfile>
                    <ParameterProfile RangeType="Interval" DataType="Integer">
                        <Name>Z</Name>
                        <RangeValues>
                            <Interval min="0" max="3"/>
                        </RangeValues>
                    </ParameterProfile>
                </Parameters>
                <Methods>
                    <MethodID>21</MethodID>
                    <MethodID>22</MethodID>
                </Methods>
            </ParameterProfileSet>
            <Method ProcessType="Atomic" ID="21">
                <SetupID>1</SetupID>
            </Method>
            <Method ProcessType="Atomic" ID="22">
                <SetupID>2</SetupID>
            </Method>
            <AverageCost>20</AverageCost>
        </MServiceImplementation>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="ServiceB" Ontology="LegoBlocks"
                          Category="Assembly" Taxonoy="none"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
                <Description>Assembles a block of size 2x4</Description>
            </MServiceType>
            .
            .
            .
        </MServiceImplementation>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="ServiceC" Ontology="LegoBlocks"
                          Category="Assembly" Taxonomy="none"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
                <Description>Assembles a block of size 2x6</Description>
            </MServiceType>
            .
            .
            .
        </MServiceImplementation>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="SupplyBase" Ontology="LegoBlocks"
                          Category="Supply" Taxonomy="sometax"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
            <Description>Supply Lego Base of a certain color to Pallet</Description>
            .
            .
            .
        </MServiceImplementation>
        <!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
    </Service_Offer>
</ResourceHolon>
```


Figure 98 Déclaration XML d'un poste de travail

Déclaration des ressources de transport

La Figure 99 présente pour illustration le fichier XML généré et transmis au *SoHMS* pour la déclaration d'une ressource de transport, qui s'applique pour les Conveyeurs, Feeders et Echangeurs. Les capacités de routage sont déclarées à travers des *ProfParamSets*. On indique ici pour chaque port d'entrée les possibles destinations (dans cet exemple, on a fait un set par couple StartPort – FinalPort).


```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RouterRH Name="Exchanger1" Technology="Exchanger" Category="Transport">
    <Description>An Exchanger between Conveyors</Description>
    <InputPorts>
        <Port>Port1.1</Port>
        <Port>Port1.3</Port>
    </InputPorts>
    <OutputPorts>
        <Port>Port1.2</Port>
        <Port>Port1.4</Port>
    </OutputPorts>
    <Service_Offer>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
        <MServiceImplementation>
            <MServiceType Name="TransportPallet" Ontology="Transport"
                          Category="Transport" Taxonomy="none"
                          ContactInfo="francisco.gamboa@univ-nantes.fr">
            <Description>Takes a pallet from an Initial Port to a
            Final Port in the System's Layout Map</Description>
        </MServiceType>
            <ParameterProfileSet>
                <Parameters>
                    <ParameterProfile RangeType="Catalogue" DataType="String">
                        <Name>StartPort</Name>
                        <RangeValues>
                            <Unit value="Port1.1"/>
                        </RangeValues>
                    </ParameterProfile>
                     <ParameterProfile RangeType="Catalogue" DataType="String">
                        <Name>FinalPort</Name>
                        <RangeValues>
                            <Unit value="Port1.2"/>
                        </RangeValues>
                    </ParameterProfile>
                </Parameters>
                <Methods>
                    <MethodID>1</MethodID>
                </Methods>
            </ParameterProfileSet>
            <!--::...............................:-->
             <ParameterProfileSet>
                ...
                        <Name>StartPort</Name>
                        ...
                            <Unit value="Port1.1"/>
                        ...
                        <Name>FinalPort</Name>
                        ...
                            <Unit value="Port1.4"/>
                        ...
            </ParameterProfileSet>
             <!--::...............................:-->
             <ParameterProfileSet>
                ...
                        <Name>StartPort</Name>
                        ...
                            <Unit value="Port1.3"/>
                        ...
                        <Name>FinalPort</Name>
                        ...
                            <Unit value="Port1.2"/>
                        ...
            </ParameterProfileSet>
             <!--::...............................:-->
             <ParameterProfileSet>
                ...
                        <Name>StartPort</Name>
                        ...
                            <Unit value="Port1.3"/>
                        ...
                        <Name>FinalPort</Name>
                        ...
                            <Unit value="Port1.4"/>
                        ...
            </ParameterProfileSet>
             <!--::...............................:-->
            <Method ProcessType="Atomic" ID="1">
                <SetupID>1</SetupID>
            </Method>
            <Method ProcessType="Atomic" ID="2">
                <SetupID>1</SetupID>
            </Method>
            <Method ProcessType="Atomic" ID="3">
                <SetupID>2</SetupID>
            </Method>
            <Method ProcessType="Atomic" ID="4">
                <SetupID>2</SetupID>
            </Method>
        </MServiceImplementation>
<!--::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::-->
    </Service_Offer>
</RouterRH>
```


Figure 99 Déclaration XML d'une ressource de transport

6.2.4 Déclaration du plan de système

On déclare le plan du système de transfert sous la forme d'un automate. La Figure 100 présente la configuration exacte de notre ligne, et la Figure 101 présente la description textuelle de l'automate en découlant.

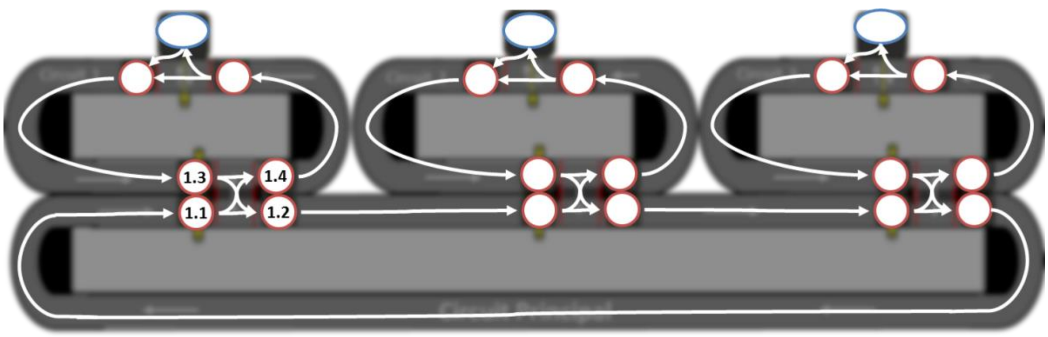


Figure 100 Plan du système de transfert

```
Automate representant le Plan de la  Chaine.

Origin; Destination; Label; Cost
-----------------------------------------
Cost units = 30
-----------------------------------------
Port3.2;Port1.1;Port1.1;4
Port1.1;Port1.2;Port1.2;4
Port1.1;Port1.4;Port1.4;4
Port1.4;Port1.OUT;Port1.OUT;4
Port1.OUT;Port1.IN;Port1.IN;4
Port1.IN;Port1.6;Port1.6;4
Port1.6;Port1.3;Port1.3;4
Port1.3;Port1.4;Port1.4;4
Port1.3;Port1.2;Port1.2;4
Port1.2;Port2.1;Port2.1;4
Port2.1;Port2.2;Port2.2;4
Port2.1;Port2.4;Port2.4;4
Port2.4;Port2.OUT;Port2.OUT;4
Port2.OUT;Port2.IN;Port2.IN;4
Port2.IN;Port2.6;Port2.6;4
Port2.6;Port2.3;Port2.3;4
Port2.3;Port2.4;Port2.4;4
Port2.3;Port2.2;Port2.2;4
Port2.2;Port3.1;Port3.1;4
Port3.1;Port3.2;Port3.2;4
Port3.1;Port3.4;Port3.4;4
Port3.4;Port3.OUT;Port3.OUT;4
Port3.OUT;Port3.IN;Port3.IN;4
Port3.IN;Port3.6;Port3.6;4
Port3.6;Port3.3;Port3.3;4
Port3.3;Port3.4;Port3.4;4
Port3.3;Port3.2;Port3.2;4
```

Figure 101 Automate représentant le plan du système de transfert

6.3 INTERACTIONS DU COMPORTEMENT REACTIF

6.3.1 Lancement des ordres de production

Le lancement des ordres de production se fait à travers l'HMI. L'HMI envoie les ordres de production avec la gamme de production au *SoHMS*, encapsulés dans un message au travers d'une connexion Socket (Figure 102).Le *SoHMS* reçoit le message, analyse le message et lance le protocole indiqué, ici le protocole « *Order Launching* ». Un Gestionnaire d'Ordre (« *Order Manager* ») est alors créé. A partir de la taille de l'ordre et le nombre maximal de produits fabriqués en parallèle indiqué par l'utilisateur (dans cette application, ce choix est laissé à l'utilisateur), le gestionnaire attribue des produits aux palettes et lance leur production. Il envoie une confirmation de retour au client via l'HMI que l'ordre a bien été lancé. À son aboutissement, chaque produit notifie au gestionnaire. Ce dernier relance des produits jusqu'à la conclusion de l'ordre ou jusqu'à ce que le nombre de palettes en parallèle maximal soit atteint.

Le *Holon Produit*, une fois lancé, démarre son comportement d'exécution en générant un nouveau fil d'exécution se séparant du fil du protocole « *Order launching* ». Le comportement implémenté dans cette première application est un comportement myopique, où l'on ne fait pas d'ordonnancement des tâches mais uniquement la planification du prochain *MService* à exécuter sur le prochain poste de travail ainsi que le chemin de transport pour y arriver. L'exécution de ce chemin est fait à la volée, en faisant des requêtes de services attribués en fonction de la disponibilité des ressources.

A la conclusion de tous les produits de l'ordre de production, le dernier fil d'exécution envoie au travers du gestionnaire un message de conclusion au client via l'HMI.

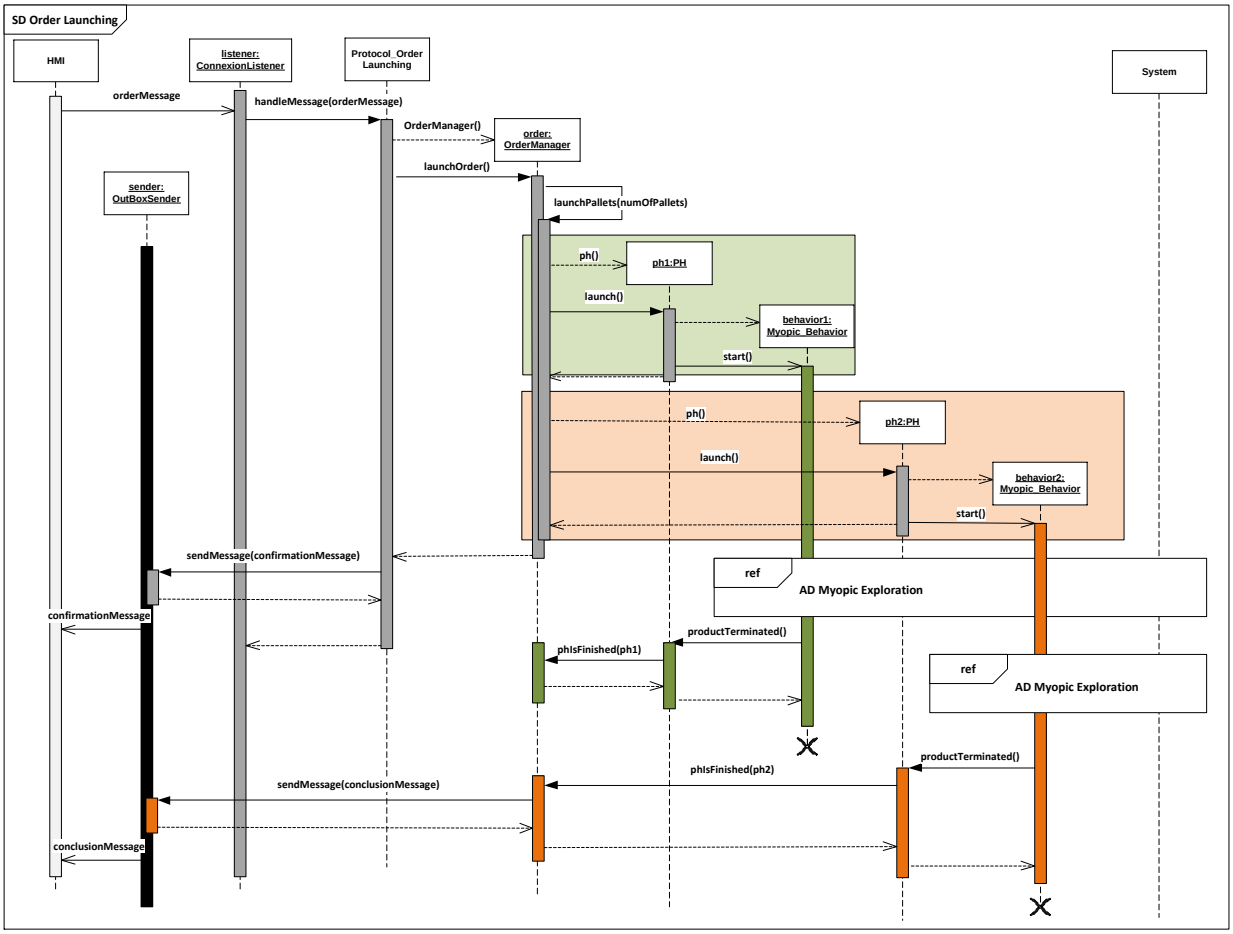


Figure 102 Lancement d'un ordre de production

6.3.2 Comportement myopique

Comme illustré Figure 103 , le *HP*, au travers de son comportement myopique, lance l'exécution d'un *Holon Ordre-Produit* (*HOP*), celui qui se chargera de mener l'exécution des plans faits par le *Holon Produit* (*HP*). Ce *HOP* se lance dans un nouveau fil d'exécution et s'active à chaque fois que le *HP* notifie la création des nouveaux plans à exécuter.

Le comportement commence par le calcul d'un plan de production à partir de l'état du produit et de sa position dans l'atelier. Ce calcul est déterminé par le protocole « *GetNext PathArcs* », qui retourne une liste de solutions faisables que le *HOP* pourra parcourir jusqu'à l'arrivée au prochain poste de travail et l'exécution du prochain service indiqué par la gamme. Les alternatives sont classées et ajoutées aux plans sous forme de graphe de solutions faisables. Le *HP* notifie alors le *HOP* de la présence de plans. Le *HOP* se réactive et démarre l'exécution des plans, dans ce cas l'exécution d'un pas de production : acheminement et exécution du *MService*.

Cette procédure se répète pour chaque étape du processus de production jusqu'à la conclusion de la gamme de production. Le *HP* planifie des plans pour le « dispatch » du produit suivant la même procédure. Le *HP* notifie le gestionnaire d'ordres de son aboutissement.

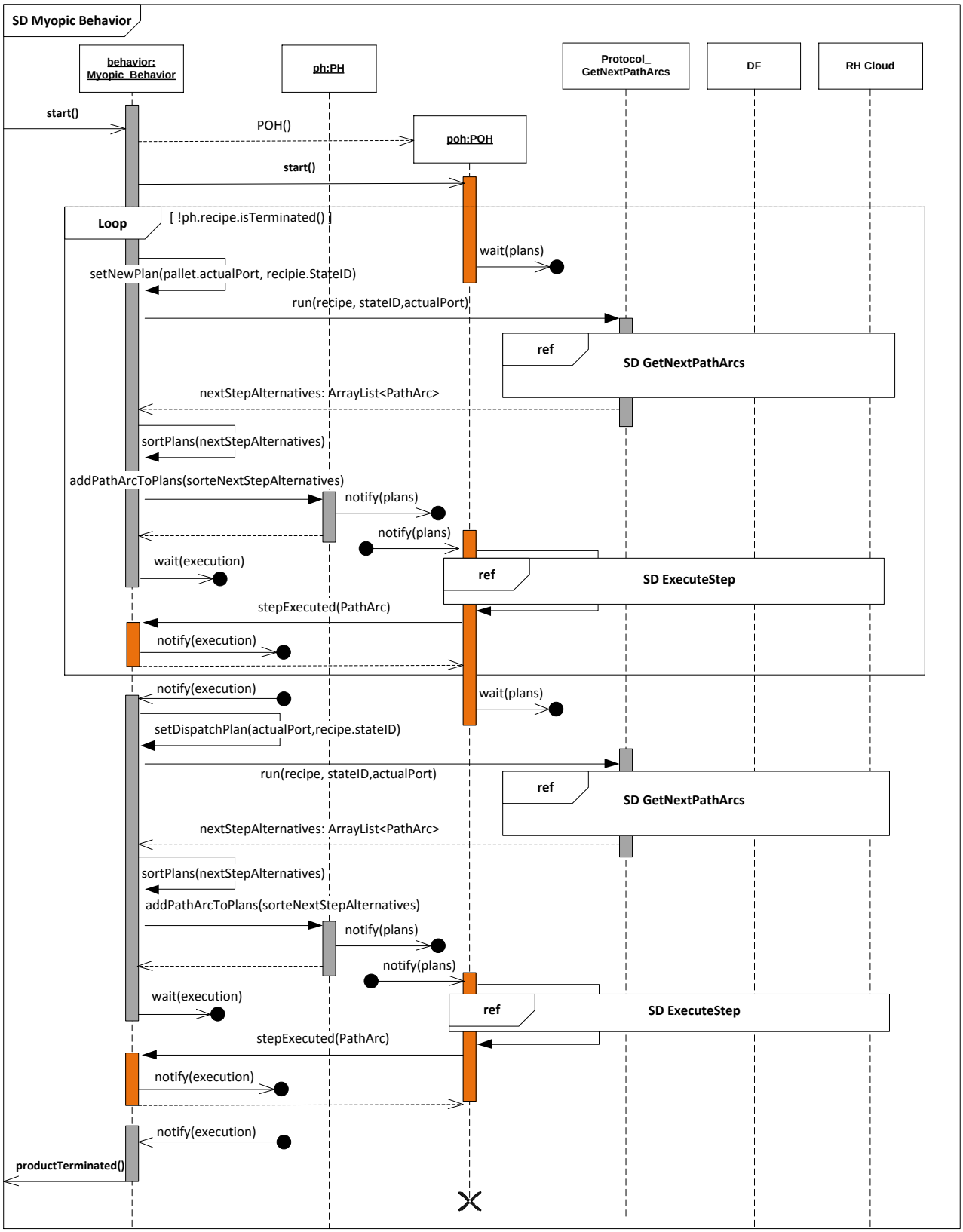


Figure 103 Comportement myopique du produit

6.3.3 Invocation réactive des MServices

La Figure 104 montre un scénario, dans le processus d'exécution du *HOP*, de requête de service de transport à l'échangeur 1. Le scénario est décrit par une palette qui est positionnée à l'entrée de

l'échangeur EX1 et lance au HR de cette ressource une requête d'exécution de service pour le service de transport entre les ports Port1.1 et le Port1.2.

Suite à la demande, EX1 doit vérifier si le port de destination sera disponible pour ainsi éviter l'exécution d'un service qui ne pourra aboutir. Pour ceci, le EX1 demande au DF la liste des propriétaires du port de destination. Le DF retourne le Conveyeur0.1 comme le propriétaire. EX1 fait alors une requête de permis de port au convoyeur 0.1. Si la réponse est positive, alors la requête de service est acceptée et le HR commande au *SIL* l'exécution du service au niveau terrain. Le *SIL* transforme alors la requête de service en message transmis par socket au MLM (contenant les algorithmes de contrôle de l'opération) pour commander l'opération. LLM envoie à son tour les signaux d'activation des actionneurs et récupère les signaux des capteurs.

A la conclusion de la tâche, le *SIL* retourne un message de confirmation. Le résultat est alors communiqué au *HOP*, qui fait alors évoluer le plan et met à jour la position de la palette.

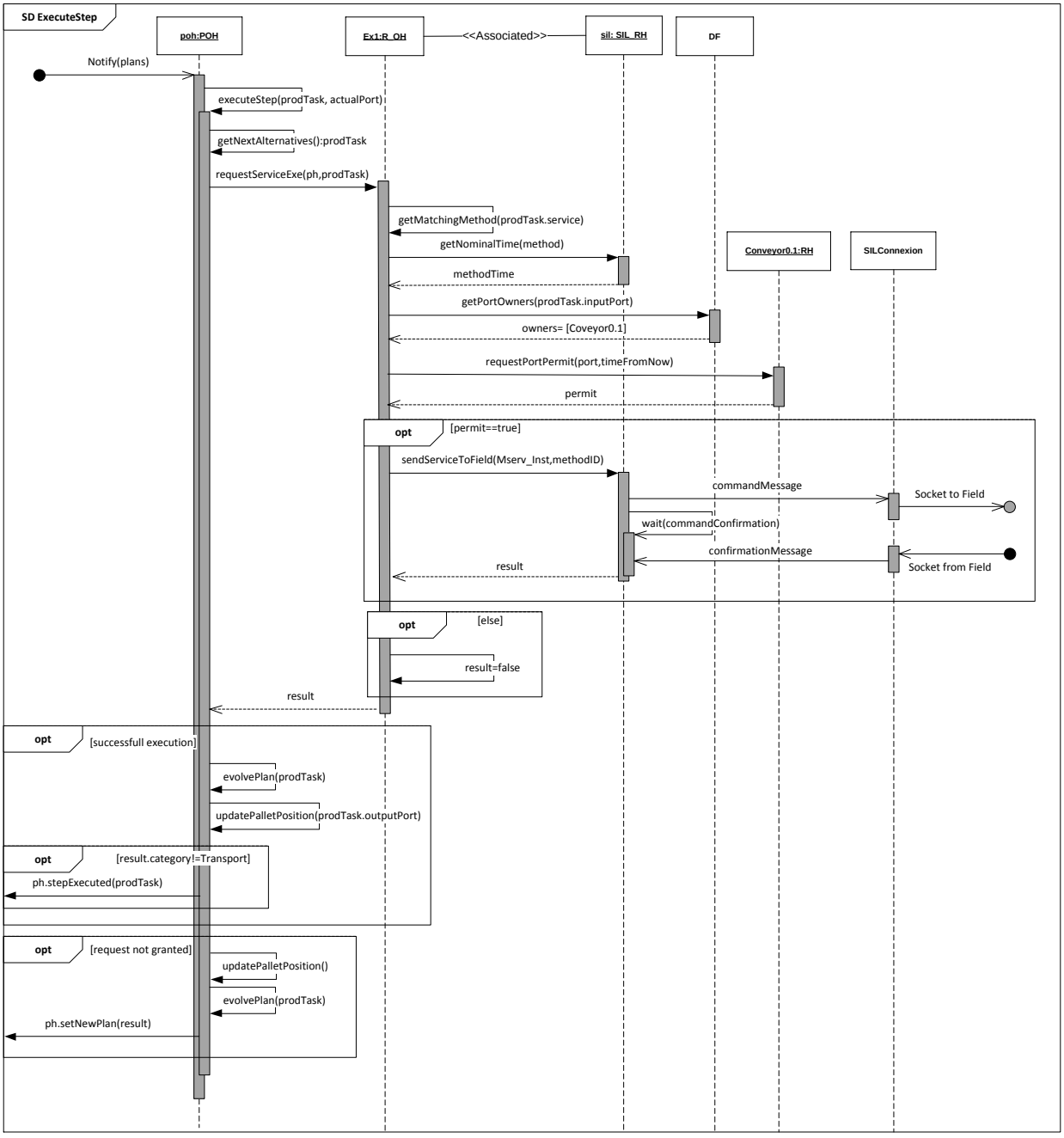


Figure 104 Scénario d'exécution d'un service de transport par l'échangeur Ex1 vers Conveyor1.1 dans le contexte de l'exécution d'un pas de production.

6.3.4 Exploration d'alternatives de production

La Figure 105 illustre le protocole pour déterminer les solutions de production pour le prochain pas de production. Il utilise la gamme flexible pour le calcul des alternatives de services (planification de processus) et la construction d'arcs pour s'ajouter au graphe de faisabilité (calcul des arcs = route de transport + MService/HR/port).

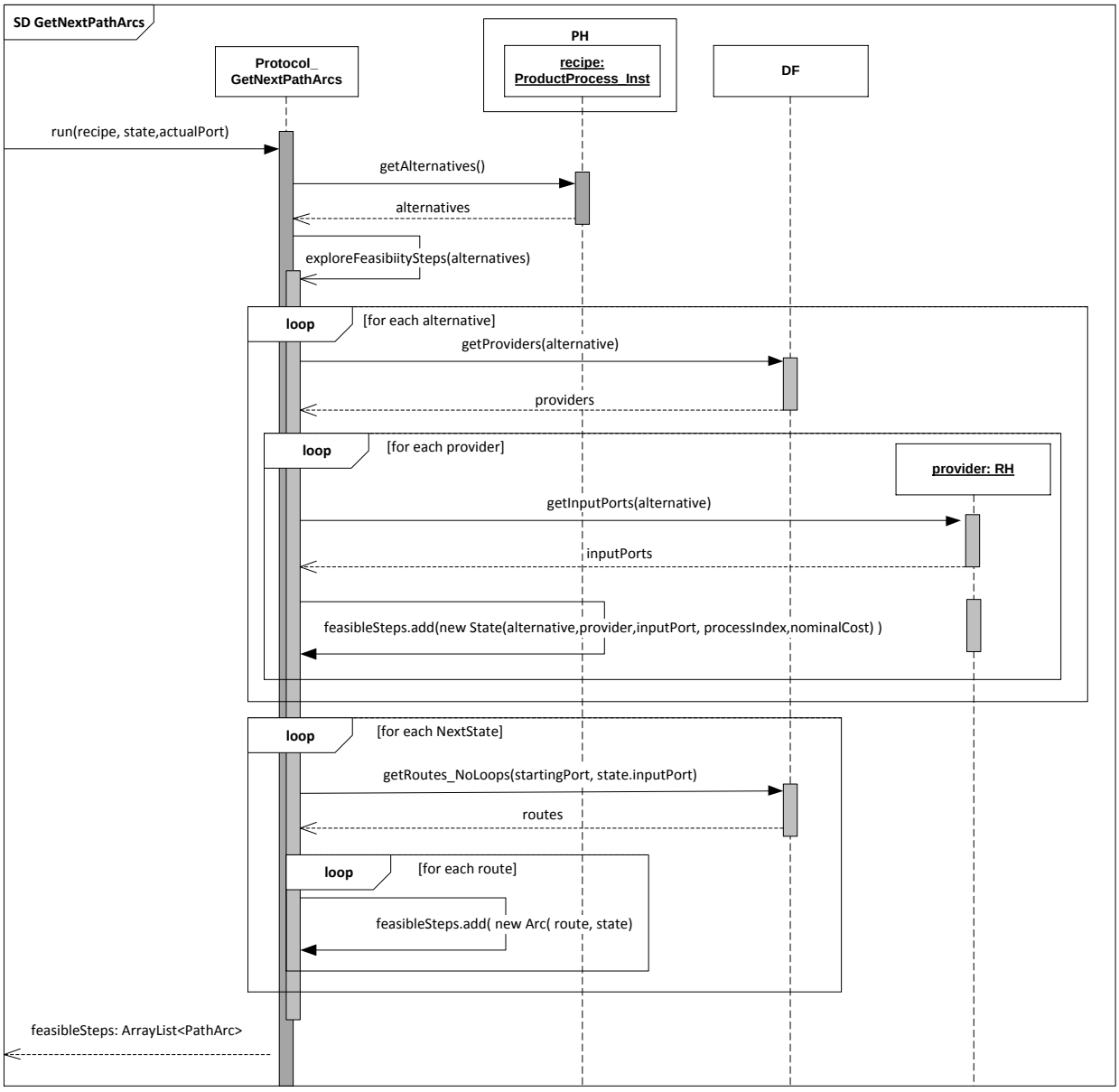


Figure 105 Protocole pour déterminer les solutions de production pour le prochain pas de production.

6.3.5 Invocation de la « Service Interface Layer »

La Figure 106 montre un scénario d'invocation de l'exécution d'un service d'assemblage de bloc (ServiceA). Dans cette application, tous les services de l'ontologie de Lego ont été implémentés sous la forme de *Processus-Dispositif* : on a donné au *SIL* la logique d'exécution de ces services comme une séquence d'actions conditionnés par une séquence de notifications. Ainsi, comme indiqué dans le diagramme, le *SIL* se charge de mener l'exécution de cette séquence puis d'en retourner le résultat

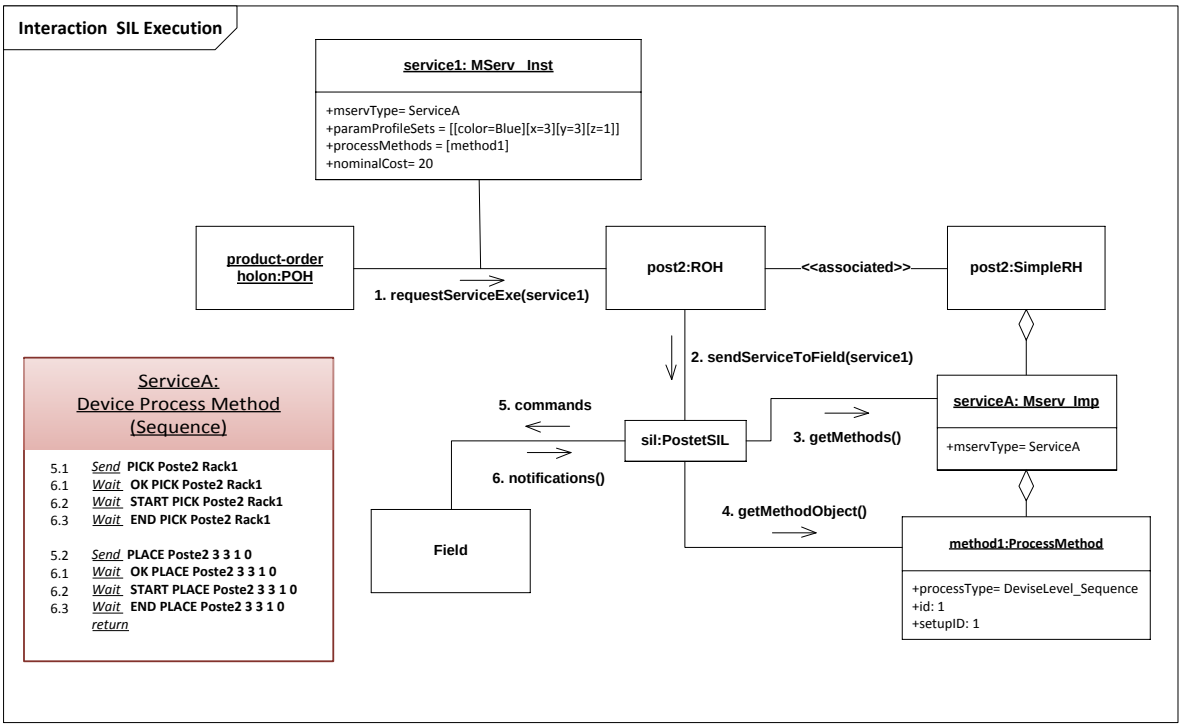


Figure 106 Scénario d'invocation de l'exécution du service d'assemblage de bloc ServiceA.

6.4 INTEGRATION DES DIFFERENTS COMPORTEMENTS

Il peut exister des ressources ayant des comportements différents selon leur nature. Il est alors possible de créer une bibliothèque de comportements par rapport à la catégorie de ressources et pour des scénarios différents. Quelques comportements sont illustrés dans cette section.

6.4.1 Comportement « Reactive Router »

Le *Reactive Router Behavior* (Figure 107) lance un coordinateur des actions dans un fil d'exécution séparé. Le *Reactive Router Behavior* reçoit des requêtes et les ajoute à une boite de requêtes. Dans cette boite, on peut trouver plusieurs requêtes à la fois, signifiants (comme pour les échangeurs par exemple) la présence de palettes dans tous ses ports d'entrée. Le *RouterPlanner* coordine alors les actions pour réduire les actions improductives. Lui aussi a la possibilité d'invoquer l'exécution d'actions par défaut.

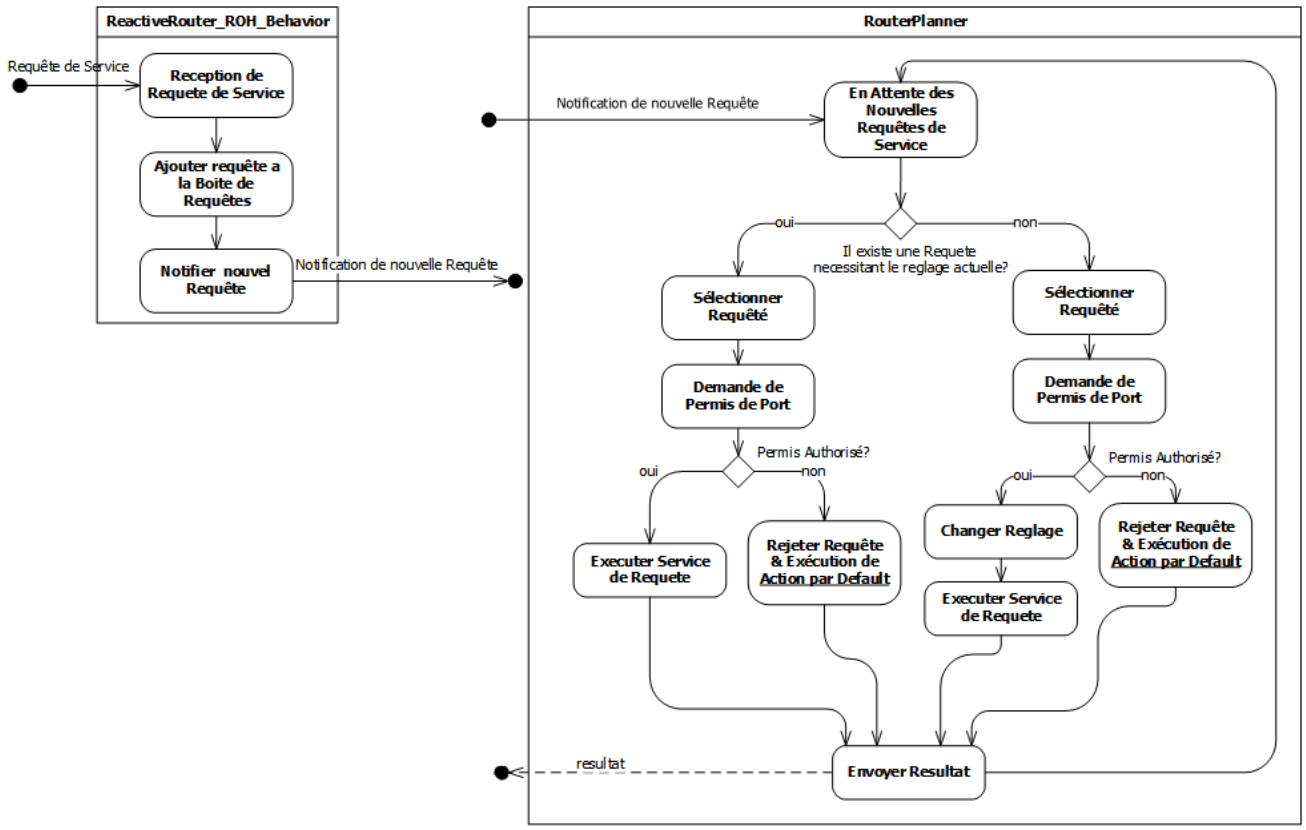


Figure 107 Comportement de routage réactif.

6.4.2 Comportement « Reactive Buffer »

Le *Reactive Buffer* (Figure 108) est une ressource passive sans capacité de raisonnement. De ce fait, toute requête de service est automatiquement acceptée. Néanmoins, il a la capacité de décision pour accorder des permis de port en fonction de sa capacité finie.

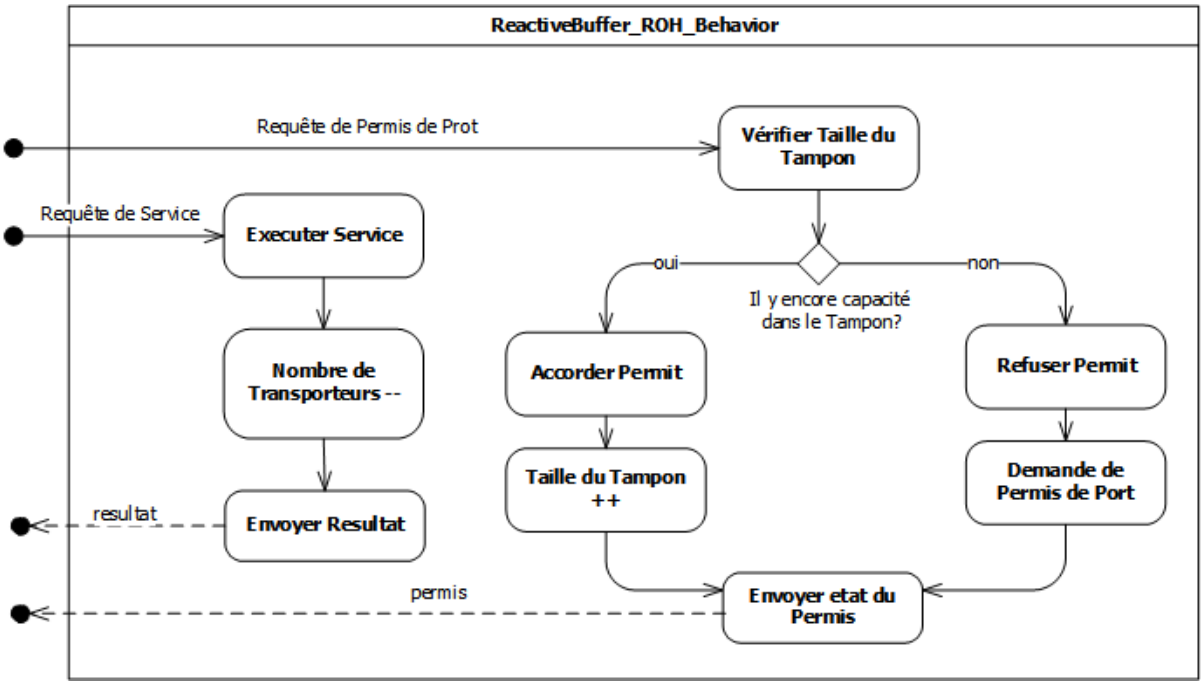


Figure 108 Comportement du tampon réactif.

6.4.3 Comportement Prédictif-Réactif «Simple Resource »

Ce comportement (Figure 109) peut être considéré standard car il s'applique à toute machine ou poste avec capacité de traitement d'un seul produit à la fois. Ce comportement a été conçu pour interagir dans un environnement réactif et prédictif. Lorsque la ressource reçoit une requête de service, le HR regarde si cette requête est comprise dans le planning de réservation.

Si elle l'est, alors le HR l'exécute. Sinon, alors le HR évalue s'il peut l'exécuter sans rentrer en conflit avec le planning :

- Evaluation du temps de traitement : Ce temps comprend le temps d'exécution du service requis, un temps de négociation à régler pour permettre au processus de requête des permis de port ou bien pour permettre au produit de tenter une autre alternative (il peut y avoir plusieurs ressources ayant le même port d'entrée) ;
- S'il y a du temps pour tout, alors une exécution normale se déroule ;
- S'il n'y a pas assez de temps pour le service, mais encore du temps pour négocier, alors la requête est juste rejetée, permettant au HP de tenter une autre solution (il reviendra s'il ne trouve pas) ;
- S'il n'y a plus de temps, alors l'action par défaut est forcée sur le produit. Le temps de négociation doit donc comprendre le temps d'exécution de la tâche par défaut.

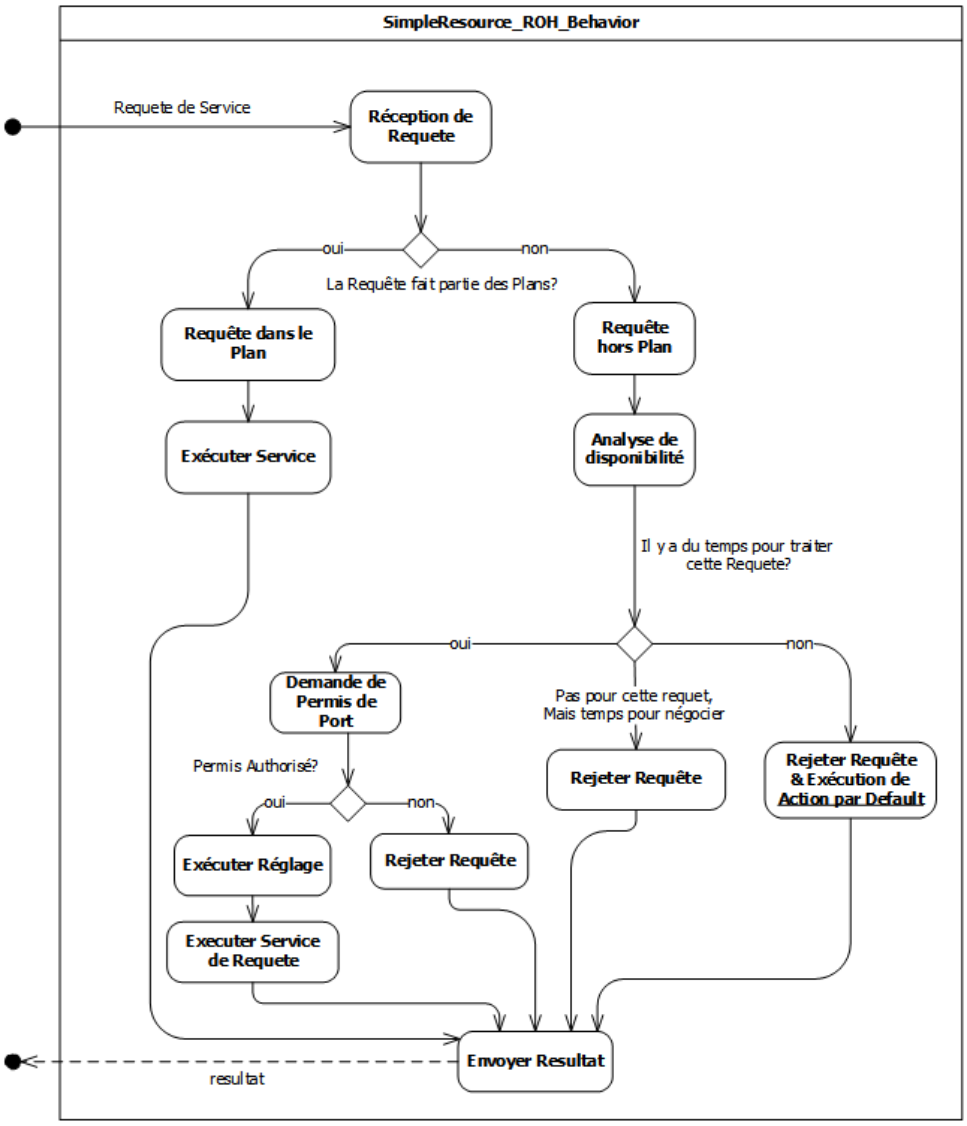


Figure 109 Comportement prédictif/réactif simple d'une ressource.

6.5 CONCLUSION

Ce dernier chapitre avait pour objectif de présenter l'implémentation du framework général de conception d'un *SoHMS* sur une ligne automatisée réelle. Après avoir défini l'ontologie d'application et les produits fabriqués sur le système, nous avons présenté l'instanciation du modèle général de *SoHMS* à notre cas d'application. Nous avons ensuite détaillé l'implémentation par la présentation des interactions mises en place entre les différents acteurs. Enfin, nous avons présenté les règles de comportement basiques actuellement implémentées.

Ces règles de comportement ne satisfont pas encore totalement à l'objectif d'orchestration indiquée dans le Chapitre 5. Toutefois, de nombreux développements sont encore en cours sur ce système et l'implémentation de ce mécanisme basé sur la *stigmergie* sera rapidement déployée. L'objectif à court terme est de pouvoir utiliser la base de programmation résultant de cette thèse pour développer de nouvelles stratégies et pouvoir valider leur performance sur ce cas d'application.

Conclusions et Perspectives

L'émergence des nouvelles architectes de contrôle distribuées répond à un besoin des entreprises d'augmenter leur réactivité et leur flexibilité de reconfiguration. Ce besoin est rendu nécessaire à cause d'un marché en constante évolution, où le produit devient de plus en plus personnalisé. Dans le domaine de la production, les architectures holoniques ont été un grand progrès pour intégrer de la flexibilité dans la structure de contrôle, grâce à une décomposition et une répartition des responsabilités de production entre entités intelligentes appelées Holons. De ce fait, les HMS accueillent très bien le paradigme de Systèmes Contrôlés par le Produit, qui est de plus en plus nécessaire dans le contexte des produits personnalisés, nécessitant des mécanismes et technologies d'identification et de traçabilité. De plus, les technologies d'implémentation des HMS, comme les Systèmes Multi-Agents par exemple, facilitent la conception des systèmes complexes, notamment avec une perspective ayant le produit comme coordinateur de la production. Issues du domaine de l'informatique, les architectures SoA sont apparues pour répondre au besoin d'interopérabilité, notamment au sein des systèmes distribués et pour introduire une flexibilité au niveau des processus. Un apport qui nous semble important est de voir le concept de service comme une abstraction et un moyen d'encapsulation des processus sous forme d'interface permettant une interopérabilité augmentée et une flexibilité lors de la construction des processus.

Mais l'apport le plus important des architectures SoA, que nous avons pu voir dans le premier chapitre de cette thèse, est le potentiel d'intégration du concept de services issu de SoA dans les HMS, ce qui s'est révélé être une bonne solution aux problèmes éventuels d'interopérabilité, non traités originellement par les HMS. Une telle architecture présente un grand potentiel de flexibilité pour les différentes fonctionnalités de contrôle, notamment pour les activités de planification et d'ordonnancement.

Dans le deuxième chapitre, nous avons proposé une adaptation de ce concept de Service (originalement conçu pour le domaine de l'informatique) pour répondre aux besoins de représentation du domaine de la production, vu comme une abstraction et description des opérations exécutées au sein d'atelier. Pour cela, nous avons proposé le modèle de *MServices,* contenant toutes les informations nécessaires à la représentation des opérations de production, notamment pour la spécification des processus de production des produits et pour la spécification des capabilités des ressources. La caractéristique la plus importante d'un tel modèle est sa capacité à représenter les structures fractales et récursives des produits (à la base du développement des familles de produits) et des ressources (à la base du principe des HMS).

Dans le troisième chapitre, nous avons alors proposé (en nous basant sur le concept de MService) une architecture Holonique inspirée des concepts et des architectures les plus reconnues dans la communauté HMS comme PROSA, HCBA et ADACOR. Cette architecture se décline comme une méthodologie de modélisation et de conception des systèmes holoniques orientés-services, où le *MService* devient le principal élément de description et d'échange au sein des interactions holoniques du système. Dans ce chapitre, nous avons redéfini les responsabilités et comportements des acteurs du système donnant au *Holon Produit* un rôle plus actif comme planificateur de sa production. Dans cette architecture, nous avons identifié et séparé les activités de planification de celles d'exécution pour prendre en compte les différentes contraintes dans le temps d'exécution et les différences de temps de calcul (les activités de planification nécessitent un grand nombre de calculs durant l'exploration des solutions).

Pour apporter de la flexibilité au niveau des processus, nous avons proposé dans le quatrième chapitre un modèle de Gammes Flexibles, basé sur le modèle de MServices, dans le but d'apporter aux mécanismes de planification et d'ordonnancement des informations sur la flexibilité séquentielle inhérente des processus et ainsi explorer un plus large espace de solutions, permettant de découvrir des solutions potentiellement plus performantes. La modélisation proposée, utilisant le formalisme de RdP (et une extension des arcs), permet la représentation des contraintes de précédence. De plus, cette modélisation est compacte et de compréhension facile pour une édition manuelle. Pour accompagner cette méthodologie, nous avons proposé un ensemble de règles de modélisation pour la génération automatique du RdP des gammes à partir d'une simple table de précédences. Une caractéristique importante de ce modèle est qu'il ne génère que des solutions faisables à chaque étape d'évolution de la gamme, ce qui est un grand avantage lors de la planification de la production en ligne.

Le cinquième chapitre nous a permis de proposer un protocole d'orchestration des processus de production utilisant les concepts présentés dans cette thèse (architecture, *MService,* gamme flexible) et utilisant une méthode du type « *Integrated Process Planning and Scheduling* » afin d'améliorer la flexibilité et l'efficacité des taches de planification et d'ordonnancement. Ce protocole se base sur une approche distribuée de génération dotée d'une vision à court terme du comportement futur du système. De plus, nous avons proposé une stratégie d'exploration de l'espace de solutions faisables permettant à terme de gérer l'explosion combinatoire due à l'exploitation des gammes flexibles.

Le dernier chapitre de ce document a été consacré à la mise en œuvre des concepts présentés dans un système expérimental réel de production. Cette implémentation a permis de montrer la faisabilité de l'approche et a abouti à la conception d'une base logicielle pérenne permettant des développements futurs de nouvelles stratégies d'exploration de l'espace de solutions.

Les perspectives de cette thèse portent sur un large nombre de domaines. Au sujet de l'intégration des services à la production, on peut étudier l'utilisation de la technologie DPWS (Device Profile for Web Services) pour la création d'un module de communication, notamment au sein des systèmes avec des dispositifs mobiles embarqués. Son implémentation permettra de profiter des protocoles standardisés pour la découverte et l'adressage des ressources et services. Les *MServices* pourront être implémentés sur une couche de plus haut niveau (couche application).

Au sujet de la modélisation de Processus de Production, dans cette thèse nous avons étudié et modélisé les caractéristiques des Processus-Produits présentés dans le chapitre 2. Une perspective porte sur la modélisation formelle des *ProcesDisp-S*, qui se caractérisent par des relations plus étroites entre les services qui les composent. Ce travail doit porter sur l'identification des contraintes et des relations entre services concurrents, sur les formalismes existants, ainsi que sur une évaluation de leur adaptation aux processus orientés-services.

Quant à l'architecture du *SoHMS*, une perspective importante porte sur le développement d'un modèle formel du module *SIL* (Service Interface Layer). Une perspective à court terme porte sur la création des modules *SIL* relatifs aux normes de réseaux de terrain classiques telles que MODBUS, PROFIBUS, etc. Lors de son développement, une question importante sera de savoir s'il faut lui conférer des capacités d'intelligence. Par exemple, le *SIL* peut servir de coordinateur des *ProcesDisp-S*.

Au sujet de la planification et de l'ordonnancement, on trouve plusieurs perspectives. La plus importante à court terme porte sur la conclusion de la mise en œuvre commencée dans le chapitre 6 des stratégies présentées dans le chapitre 5. Après leur mise en œuvre, une étude sera réalisée pour évaluer le comportement émergent résultant de ces stratégies. Une telle évaluation devra mesurer la stabilité (la convergence des explorations vers des plans de production), la performance (étude comparative utilisant des benchmarks et des comparaisons avec d'autres architectures), la réactivité (capacité à trouver toujours

une solution au moment pertinent), la demande des ressources de calcul (trafic des messages, la quantité de mémoire virtuelle requise pour l'exploration, parallélisme des opérations de calcul) et leur efficacité (nombre d'états explorés, réduction des efforts dupliqués).

A plus long terme, une perspective serait de proposer une plateforme pour un MES holonique permettant un développement unifié et collaboratif. Celle-ci serait conçue de façon modulaire pour permettre d'intégrer facilement de nouvelles solutions ou développements provenant de différentes sources de recherche académique. Cette plateforme permettrait de tester et comparer nos propositions sur les différents équipements expérimentaux disponibles dans la communauté, notamment en utilisant les benchmarks de la littérature. Le développement de cet outil s'inscrit dans un objectif général visant à définir une structure logicielle générique facilitant le transfert de ces concepts et théories dans le milieu industriel.

Références

Adam, E., Zambrano, G., Pach, C., Berger, T., Trentesaux, D., 2011. Myopic Behaviour in Holonic Multiagent Systems for Distributed Control of FMS, in: Trends in Practical Applications of Agents and Multiagent Systems, Advances in Intelligent and Soft Computing. Springer, pp. 91–98.

Babiceanu, R.F., Chen, F.F., 2006. Development and Applications of Holonic Manufacturing Systems: A Survey. J. Intell. Manuf. 17, 111–131. doi:10.1007/s10845-005-5516-y

Baker, N., 2005. ZigBee and Bluetooth strengths and weaknesses for industrial applications. Comput. Control Eng. J. 16, 20–25. doi:10.1049/cce:20050204

Barbosa, J., Leitão, P., Adam, E., Trentesaux, D., 2015. Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution. Comput. Ind. 66, 99–111. doi:10.1016/j.compind.2014.10.011

Belle, J.V., Germain, B.S., Verstraete, P., Valckenaers, P., Ali, O., Brussel, H.V., Cattrysse, D., 2009. A Holonic Chain Conveyor Control System: An Application, in: Mařík, V., Strasser, T., Zoitl, A. (Eds.), Holonic and Multi-Agent Systems for Manufacturing, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 234–243.

Bellifemine, F.L., Caire, G., Greenwood, D., 2007. Developing Multi-Agent Systems with JADE, 1 edition. ed. Wiley, Hoboken, NJ.

Berger, T., Deneux, D., Bonte, T., Cocquebert, E., Trentesaux, D., 2015. Arezzo-flexible manufacturing system: A generic flexible manufacturing system shop floor emulator approach for high-level control virtual commissioning. Concurr. Eng. 1063293X15591609. doi:10.1177/1063293X15591609

Blanc, P., 2006. Pilotage par approche holonique d'un système de production de vitres de sécurité feuilletées. Nantes.

Blanc, P., Demongodin, I., Castagna, P., 2008. A holonic approach for manufacturing execution system design: An industrial application. Eng. Appl. Artif. Intell. 21, 315–330.

Bohn, H., Bobek, A., Golatowski, F., 2006. SIRENA-Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains, in: Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on. IEEE, pp. 43–43.

Borangiu, T., Raileanu, S., Trentesaux, D., Berger, T., Iacob, I., 2014. Distributed manufacturing control with extended CNP interaction of intelligent products. J. Intell. Manuf. 25, 1065–1075. doi:10.1007/s10845-013-0740-3

Borgo, S., Leitão, P., 2004. The Role of Foundational Ontologies in Manufacturing Domain Applications, in: Meersman, R., Tari, Z. (Eds.), On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 670–688.

Bourdeaud'huy, T., Toguyeni, A., 2006. Analysis of Reconfiguration Strategies Based On Petri Nets Models and Optimization Techniques, in: 2006 8th International Workshop on Discrete Event Systems. Presented at the 2006 8th International Workshop on Discrete Event Systems, pp. 319–324. doi:10.1109/WODES.2006.382526

Cândido, G., Barata, J., Colombo, A.W., Jammes, F., 2009. SOA in reconfigurable supply chains: A research roadmap. Eng. Appl. Artif. Intell., Artificial Intelligence Techniques for Supply Chain Management 22, 939–949.

Cardin, O., Castagna, P., 2009. Using online simulation in Holonic Manufacturing Systems. Eng. Appl. Artif. Intell. 22, 1025–1033.

Cardin, O., Mebarki, N., Pinot, G., 2013. A study of the robustness of the group scheduling method using an emulation of a complex FMS. Int. J. Prod. Econ. 146, 199–207. doi:10.1016/j.ijpe.2013.06.023

Chang, T.-C., Wysk, R.A., 1985. An introduction to automated process planning systems. Prentice-Hall.

Chen, Q., Khoshnevis, B., 1993. Scheduling with flexible process plans. Prod. Plan. Control 4, 333–343. doi:10.1080/09537289308919455

Child, P., Diederichs, R., Sanders, F.-H., Wisniowski, S., 1991. SMR Forum: The Management of Complexity 33, 73–80.

Chirn, J.-L., McFarlane, D.C., 2000. A holonic component-based approach to reconfigurable manufacturing control architecture, in: 11th International Workshop on Database and Expert Systems Applications Proceedings. pp. 219–223. doi:10.1109/DEXA.2000.875030

Chové, E., Castagna, P., Abbou, R., 2009. Hoist Scheduling Problem: Coupling reactive and predictive approaches, in: INCOM 2009 Proceedings. pp. 2077–2082.

Christensen, J.H., 1994. Holonic manufacturing systems: initial architecture and standards directions, in: Proc 1st Euro Wkshp on Holonic Manufacturing Systems.

Colombo, A.W., Karnouskos, S., Bangemann, T., 2014. Towards the Next Generation of Industrial Cyber-Physical Systems, in: Industrial Cloud-Based Cyber-Physical Systems. Springer, pp. 1–22.

Conway, R.W., Maxwell, W.L., Miller, L.W., 1967. Theory Of Scheduling.

Crow, K., 1992. Computer-Aided Process Planning. DRM Assoc.

Dechter, R., Pearl, J., 1985. Generalized Best-first Search Strategies and the Optimality of A*. J ACM 32, 505–536. doi:10.1145/3828.3830

Delamer, I.M., Lastra, J.L.M., 2006. Ontology Modeling of Assembly Processes and Systems using Semantic Web Services, in: 2006 IEEE International Conference on Industrial Informatics. Presented at the 2006 IEEE International Conference on Industrial Informatics, pp. 611–617. doi:10.1109/INDIN.2006.275631

Den Haan, J., 2007. SOA and Service Identification - by Johan Den Haan [WWW Document]. Enterp. Archit. URL http://www.theenterprisearchitect.eu/blog/2007/04/26/soa-and-service-identification/ (accessed 8.23.15).

De Souza, L.M.S., Spiess, P., Guinard, D., Köhler, M., Karnouskos, S., Savio, D., 2008. Socrades: A web service based shop floor integration infrastructure, in: The Internet of Things. Springer, pp. 50–67.

Deutsches Institute für Normung e. V., 2003. Manufacturing Processes Joining. DIN 8593 Parts 0-8.

Dicesare, F., Harhalakis, G., Proth, J.-M., Silva-Suarez, M., Vernadat, F., 2012. Practice of Petri Nets in Manufacturing. Springer Science & Business Media.

Du, X., Jiao, J., Tseng, M.M., 2001. Architecture of Product Family: Fundamentals and Methodology. Concurr. Eng. 9, 309–325. doi:10.1177/1063293X0100900407

Erl, T., 2005. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall, Upper Saddle River, NJ.

Fisher, M., Ramdas, K., Ulrich, K., 1999. Component Sharing in the Management of Product Variety: A Study of Automotive Braking Systems. Manag. Sci. 45, 297–315. doi:10.1287/mnsc.45.3.297

Gamboa Quintanilla, F., Cardin, O., Castagna, P., 2014a. Product Specification for Flexible Workflow Orchestrations in Service Oriented Holonic Manufacturing Systems, in: Service Orientation in Holonic and Multi-Agent Manufacturing and Robotics, Studies in Computational Intelligence. Springer, pp. 177–193.

Gamboa Quintanilla, F., Cardin, O., Castagna, P., 2013a. Evolution of a Flexible Manufacturing System: From Communicating to Autonomous Product, in: Service Orientation in Holonic

and Multi Agent Manufacturing and Robotics, Studies in Computational Intelligence. Springer, pp. 167–180.

Gamboa Quintanilla, F., Cardin, O., L'Anton, A., Castagna, P., 2016. A Petri net-based methodology to increase flexibility in service-oriented holonic manufacturing systems. Comput. Ind. 76, 53–68. doi:10.1016/j.compind.2015.09.002

Gamboa Quintanilla, F., Cardin, O., L'Anton, A., Castagna, P., 2014b. Product Specification for Flexible Workflow Orchestrations in Service Oriented Holonic Manufacturing Systems. Presented at the SOHOMA'14, Nancy –France.

Gamboa Quintanilla, F., Kubler, S., Cardin, O., Castagna, P., 2013b. Product Specification in a Service-Oriented Holonic Manufacturing System Using Petri-Nets, in: Tsuzuki, M. (Ed.), IMS 2013. São Paulo, Brazil, pp. 342–347. doi:10.3182/20130522-3-BR-4036.00094

Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., 2001. Understanding top-level ontological distinctions, in: Proceedings of IJCAI 2001 Workshop on Ontologies and Information Sharing.

Gardey, G, Lime, D, Magnin, M, Roux, OH, 2005. Roméo: A tool for analyzing time Petri nets". In 17th International Conference on Computer Aided Verification (CAV'05). Vol. 3576 Lect. Notes Comput. Sci. 418–423.

Grönroos, C., 2000. Service Management and Marketing: A Customer Relationship Management Approach. Wiley.

Gschwind, B., Ménard, L., Albuisson, M., Wald, L., 2006. Converting a successful research project into a sustainable service: the case of the SoDa Web service. Environ. Model. Softw. 21, 1555–1561.

Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., Savio, D., 2010. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. Serv. Comput. IEEE Trans. On 3, 223–235.

Hart, P.E., Nilsson, N.J., Raphael, B., 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Trans. Syst. Sci. Cybern. 4, 100–107. doi:10.1109/TSSC.1968.300136

Henrioud, J.M., Bourjault, A., 1992. Computer Aided Assembly Process Planning. Proc. Inst. Mech. Eng. Part B J. Eng. Manuf. 206, 61–66. doi:10.1243/PIME_PROC_1992_206_056_02

Henrioud, J.-M., Bourjault, A., 1991. LEGA: a computer-aided generator of assembly plans, in: Mello, L.S.H. de, Lee, S. (Eds.), Computer-Aided Mechanical Assembly Planning, The Springer International Series in Engineering and Computer Science. Springer US, pp. 191–215.

Holvoet, T., Valckenaers, P., 2006. Beliefs, Desires and Intentions Through the Environment, in: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '06. New York, NY, USA, pp. 1052–1054.

ISA-95, 2000. ISA-95.01 Models & Terminology [WWW Document]. URL http://www.isa-95.com/subpages/technology/isa-95/isa-95-01.php (accessed 5.13.14).

Jammes, F., Mensch, A., Smit, H., 2007. Service-Oriented Device Communications Using the Devices Profile for Web services, in: 21st International Conference on Advanced Information Networking and Applications Workshops, 2007, AINAW '07. Presented at the 21st International Conference on Advanced Information Networking and Applications Workshops, 2007, AINAW '07, pp. 947–955. doi:10.1109/AINAW.2007.331

Jammes, F., Smit, H., 2005. Service-oriented paradigms in industrial automation. IEEE Trans. Ind. Inform. 1, 62–70. doi:10.1109/TII.2005.844419

Jammes, F., Smit, H., Lastra, J.L.M., Delamer, I.M., 2005. Orchestration of service-oriented manufacturing processes, in: 10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005. Presented at the 10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005, p. 8 pp.–624.

Jeng, M.D., DiCesare, F., 1990. A review of synthesis techniques for Petri nets, in: , Proceedings of Rensselaer's Second International Conference on Computer Integrated Manufacturing, 1990.

Presented at the , Proceedings of Rensselaer's Second International Conference on Computer Integrated Manufacturing, 1990, pp. 348–355. doi:10.1109/CIM.1990.128124

Jiao, J. (Roger), Simpson, T.W., Siddique, Z., 2007. Product family design and platform-based product development: a state-of-the-art review. J. Intell. Manuf. 18, 5–29. doi:10.1007/s10845-007-0003-2

Jiao, J., Tseng, M.M., Duffy, V.G., Lin, F., 1998. Product family modeling for mass customization. Comput. Ind. Eng., Selected Papers from the 22nd ICC and IE Conference 35, 495–498. doi:10.1016/S0360-8352(98)00142-9

J. Wyns, H. Van Brussel, P. Valckenaers, L. Bongaerts, 1996. Workstation Architecture in holonic manufacturing systems. Proc 28th CIRP Int. Semin. Manuf. Syst. 220–231.

Kanchanasevee, P., Biswas, G., Kawamura, K., Tamura, S., 1997. Contract-net-based scheduling for holonic manufacturing systems. pp. 108–115. doi:10.1117/12.294424

Kärkkäinen, M., 2003. Increasing efficiency in the supply chain for short shelf life goods using RFID taggingnull. Int. J. Retail Distrib. Manag. 31, 529–536. doi:10.1108/09590550310497058

Koestler, A., 1968. THE GHOST IN THE MACHINE.

Komoda, N., 2006. Service Oriented Architecture (SOA) in Industrial Systems, in: 2006 IEEE International Conference on Industrial Informatics. Presented at the 2006 IEEE International Conference on Industrial Informatics, pp. xxiii–xxiii. doi:10.1109/INDIN.2006.275681

Kruth, J.P., Detand, J., 1992. A CAPP System for Nonlinear Process Plans. CIRP Ann. - Manuf. Technol. 41, 489–492. doi:10.1016/S0007-8506(07)61251-7

Kruth, J.-P., Detand, J., Van Zeir, G., Kempenaers, J., Pinte, J., 1996. Methods to Improve the Response Time of a CAPP System That Generates Non-linear Process Plans. Adv Eng Softw 25, 9–17. doi:10.1016/0965-9978(95)00081-X

Legat, C., Vogel-Heuser, B., 2015. An Orchestration Engine for Services-Oriented Field Level Automation Software, in: Borangiu, T., Thomas, A., Trentesaux, D. (Eds.), Service Orientation in Holonic and Multi-Agent Manufacturing, Studies in Computational Intelligence. Springer International Publishing, pp. 71–80.

Leitão, P., Restivo, F., 2006. ADACOR: A holonic architecture for agile and adaptive manufacturing control. Comput. Ind. 57, 121–130. doi:10.1016/j.compind.2005.05.005

Lime, D, Roux, OH, Seidner, C, Traonouez, LM, 2009. Romeo: A parametric model-checker for Petri nets with stopwatches. 15th Int. Conf. Tools Algorithms Constr. Anal. Syst. TACAS 2009 54–57.

Linthicum, D., 2012. Service Oriented Architecture (SOA), in: Microsoft MSDN Library.

Marik, V., McFarlane, D., 2005. Industrial Adoption of Agent-Based Technologies. IEEE Intell. Syst. 20, 27–35. doi:10.1109/MIS.2005.11

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K., 2004. OWL-S: Semantic Markup for Web Services [WWW Document]. URL http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/ (accessed 5.5.14).

Martinez, M.T., Favrel, J., Fhodous, P., 2000. Product Family Manufacturing Plan Generation and Classification. Concurr. Eng. Res. Appl. 12–22.

McFarlane, D.C., Bussmann, S., 2003. Holonic Manufacturing Control: Rationales, Developments and Open Issues, in: Deen, S.M. (Ed.), Agent-Based Manufacturing, Advanced Information Processing. Springer Berlin Heidelberg, pp. 303–326.

Mendes, J.M., Leitão, P., Restivo, F., Colombo, A.W., 2010. Process optimization of service-oriented automation devices based on Petri nets, in: 2010 8th IEEE International Conference on Industrial Informatics (INDIN). Presented at the 2010 8th IEEE International Conference on Industrial Informatics (INDIN), pp. 274–279. doi:10.1109/INDIN.2010.5549413

Mendes, J.M., Restivo, F., Leitao, P., Colombo, A.W., 2009. Customizable service-oriented Petri net controllers, in: 35th Annual Conference of IEEE Industrial Electronics, 2009. IECON '09.

Presented at the 35th Annual Conference of IEEE Industrial Electronics, 2009. IECON '09, pp. 4341–4346. doi:10.1109/IECON.2009.5414914

Meyer, M.H., Lehnerd, A.P., 1997. The Power of Product Platforms. Simon and Schuster.

Meyer, M.H., Tertzakian, P., Utterback, J.M., 1997. Metrics for Managing Research and Development in the Context of the Product Family. Manag. Sci. 43, 88–111. doi:10.1287/mnsc.43.1.88

Molina, A., Rodriguez, C.A., Ahuett, H., Cortés, J.A., Ramírez, M., Jiménez, G., Martinez, S., 2005. Next-generation manufacturing systems: key research issues in developing and integrating reconfigurable and intelligent machines. Int. J. Comput. Integr. Manuf. 18, 525–536. doi:10.1080/09511920500069622

Morariu, C., Morariu, O., Borangiu, T., 2013. Customer order management in service oriented holonic manufacturing. Comput. Ind. 64, 1061–1072. doi:10.1016/j.compind.2013.07.007

Morel, G., Valckenaers, P., Faure, J.-M., Pereira, C.E., Diedrich, C., 2007. Manufacturing plant control challenges and issues. Control Eng. Pract. 15, 1321–1331. doi:10.1016/j.conengprac.2007.05.005

Moritz, G., Zeeb, E., Prüter, S., Golatowski, F., Timmermann, D., Stoll, R., 2009. Devices profile for web services in wireless sensor networks: adaptations and enhancements, in: Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on. IEEE, pp. 1–8.

Murata, T., 1989. Petri nets: Properties, analysis and applications. Proc. Inst. Electr. Electron. Eng. 77, 541–580. doi:10.1109/5.24143

National Institute of Standards and Technology (NIST), 2008. Process Specification Language (PSL) [WWW Document]. URL http://www.mel.nist.gov/psl/ (accessed 5.2.14).

Pach, C., Berger, T., Bonte, T., Trentesaux, D., 2014. ORCA-FMS: a dynamic architecture for the optimized and reactive control of flexible manufacturing scheduling. Comput. Ind. 65, 706–720. doi:10.1016/j.compind.2014.02.005

Pahl, C., Zhu, Y., 2006. A Semantical Framework for the Orchestration and Choreography of Web Services. Electron. Notes Theor. Comput. Sci., Proceedings of the International Workshop on Web Languages and Formal Methods (WLFM 2005)Proceedings of the International Workshop on Web Languages and Formal Methods (WLFM 2005) 151, 3–18. doi:10.1016/j.entcs.2005.07.033

Palmer, G.J., 1994. An integrated approach to manufacturing planning: optimisation in process planning and job shop scheduling. University of Huddersfield.

Pannequin, R., 2007. Proposition d'un environnement de modélisation et de test d'architectures de pilotage par le produit de systèmes de production (phdthesis). Université Henri Poincaré - Nancy I.

Pearl, J., 1984. Heuristics: Intelligent search strategies for computer problem solving.

Perrey, R., Lycett, M., 2003. Service-oriented architecture, in: 2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings. Presented at the 2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings, pp. 116–119. doi:10.1109/SAINTW.2003.1210138

Pierreval, H., Mebarki, N., 1997. Dynamic scheduling selection of dispatching rules for manufacturing system. Int. J. Prod. Res. 35, 1575–1591. doi:10.1080/002075497195137

Pisanelli, D.M., Gangemi, A., Steve, G., 2002. Ontologies and Information Systems: the Marriage of the Century, in: In Proceedings of LYEE Workshop.

Ramirez, A.C., 2006. Contribution à la Modélisation et à la Gestion des Interactions Produit-Processus dans la Chaîne Logistique par l'Approche Produits Communicants (phdthesis). Université Henri Poincaré - Nancy I.

Rodrigues, N., Leitão, P., Oliveira, E., 2015. Self-interested Service-Oriented Agents Based on Trust and QoS for Dynamic Reconfiguration, in: Borangiu, T., Thomas, A., Trentesaux, D. (Eds.), Service Orientation in Holonic and Multi-Agent Manufacturing, Studies in Computational Intelligence. Springer International Publishing, pp. 209–218.

Roux, O.H., Lime, D., 2004. Time Petri Nets with Inhibitor Hyperarcs. Formal Semantics and State Space Computation, in: Cortadella, J., Reisig, W. (Eds.), Applications and Theory of Petri Nets 2004, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 371–390.

Sallez, Y., 2014. Proposition of an Analysis Framework to Describe the "Activeness" of a Product during Its Life Cycle, in: Service Orientation in Holonic and Multi-Agent Manufacturing and Robotics, Studies in Computational Intelligence. Springer, pp. 257–270.

Schierholt, K., 1999. Process configuration: combining the principles of product configuration and process planning, in: Mertins, K., Krause, O., Schallock, B. (Eds.), Global Production Management, IFIP — The International Federation for Information Processing. Springer US, pp. 391–398.

Schlenoff, C., Gruninger, M., Tissot, F., Valois, J., Road, T.C., Inc, S., Lubell, J., Lee, J., 1999. The Process Specification Language (PSL) Overview and Version 1.0 Specification.

Shen, W., Wang, L., Hao, Q., 2006. Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 36, 563–577. doi:10.1109/TSMCC.2006.874022

Simpson, T.W., Maier, J.R., Mistree, F., 2001. Product platform design: method and application. Res. Eng. Des. 13, 2–22. doi:10.1007/s001630100002

Smith, R.G., 1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Trans. Comput. C-29, 1104–1113. doi:10.1109/TC.1980.1675516

Sousa, P., Ramos, C., 1999. A distributed architecture and negotiation protocol for scheduling in manufacturing systems. Comput. Ind. 38, 103–113. doi:10.1016/S0166-3615(98)00112-2

Tan, W., Khoshnevis, B., 2000. Integration of process planning and scheduling— a review. J. Intell. Manuf. 11, 51–63. doi:10.1023/A:1008952024606

Toguyeni, A., 2006. Design of Modular and Hierarchical Controllers for Reconfigurable Manufacturing Systems, in: IMACS Multiconference on Computational Engineering in Systems Applications. Presented at the IMACS Multiconference on Computational Engineering in Systems Applications, pp. 1004–1011. doi:10.1109/CESA.2006.4281795

Trentesaux, D., 2009. Distributed control of production systems. Eng. Appl. Artif. Intell., Distributed Control of Production Systems 22, 971–978. doi:10.1016/j.engappai.2009.05.001

Ueda, K., 1992. A Concept for Bionic Manufacturing Systems Based on DNA-type Information, in: Proceedings of the IFIP TC5 / WG5.3 Eight International PROLAMAT Conference on Human Aspects in Computer Integrated Manufacturing. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, pp. 853–863.

Ulrich, K., Eppinger, S.D., 1995. Product Design and Development, First Edition edition. ed. Mcgraw-Hill College, New York.

Valckenaers, P., Bonneville, F., Van Brussel, H., Bongaerts, L., Wyns, J., 1994. Results of the holonic control system benchmark at KU Leuven, in: , Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology, 1994. Presented at the , Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology, 1994, pp. 128–133. doi:10.1109/CIMAT.1994.389083

Valckenaers, P., Hadeli, Saint Germain, B., Verstraete, P., Van Brussel, H., 2006. Emergent short-term forecasting through ant colony engineering in coordination and control systems. Adv. Eng. Inform., Design of Complex Adaptive Systems 20, 261–278. doi:10.1016/j.aei.2006.01.007

Valckenaers, P., Van Brussel, H., Bongaerts, L., Bonneville, F., 1995. Programming, scheduling, and control of flexible assembly systems. Comput. Ind., Computer Integrated Manufacturing and Industrial Automation 26, 209–218. doi:10.1016/0166-3615(95)00013-T

Valckenaers, P., Van Brussel, H., Hadeli, Bochmann, O., Saint Germain, B., Zamfirescu, C., 2003. On the design of emergent systems: an investigation of integration and interoperability

issues. Eng. Appl. Artif. Intell., Intelligent Manufacturing 16, 377–393. doi:10.1016/S0952-1976(03)00080-0

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., Peeters, P., 1998. Reference architecture for holonic manufacturing systems: PROSA. Comput. Ind. 37, 255–274. doi:10.1016/S0166-3615(98)00102-X

Verstraete, P., Valckenaers, P., Van Brussel, H., Saint Germain, B., Hadeli, K., Van Belle, J., 2008. Towards robust and efficient planning execution. Eng. Appl. Artif. Intell. 21, 304–314. doi:10.1016/j.engappai.2007.09.002

Vos, J.A.W.M., Kals, H.J.J., TU Delft: Design, Engineering and Production, TU Delft, Delft University of Technology, 2001. Module and System Design in Flexible Automated Assembly.

Warnecke, H.-J., 1993. The Fractal Company. Springer Berlin Heidelberg, Berlin, Heidelberg.

Wong, C.Y., McFarlane, D., Ahmad Zaharudin, A., Agarwal, V., 2002. The intelligent product driven supply chain, in: 2002 IEEE International Conference on Systems, Man and Cybernetics. Presented at the 2002 IEEE International Conference on Systems, Man and Cybernetics, p. 6. doi:10.1109/ICSMC.2002.1173319

Zambrano Rey, G., Bonte, T., Prabhu, V., Trentesaux, D., 2014. Reducing myopic behavior in FMS control: A semi-heterarchical simulation–optimization approach. Simul. Model. Pract. Theory, Simulation-Optimization of Complex Systems: Methods and Applications 46, 53–75. doi:10.1016/j.simpat.2014.01.005

Zambrano Rey, G., Pach, C., Aissani, N., Bekrar, A., Berger, T., Trentesaux, D., 2013. The control of myopic behavior in semi-heterarchical production systems: A holonic framework. Eng. Appl. Artif. Intell. 26, 800–817. doi:10.1016/j.engappai.2012.08.011

Zweben, M., Fox, M., 1994. Intelligent Scheduling. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Francisco Gamboa Quintanilla

Couplage des Architectures Holonique et Orientée-Services pour la Conception de Systèmes de Production Agiles.

Résumé

Pour atteindre des objectifs en termes de réactivité, flexibilité, coûts et productivité, les entreprises recherchent des solutions remplaçant les systèmes de contrôle manufacturiers conventionnels afin de gagner en robustesse, adaptabilité, configuration rapide et maximisation de l'utilisation des ressources. Les Architectures Holoniques et Orientées-Services ont été proposées en tant que solutions de conception pour de tels types de systèmes, le premier dans le domaine manufacturier fournissant un contrôle flexible, le dernier en informatique au niveau process. Leur combinaison a été reconnue comme une solution encore plus attractive pour la conception des systèmes de production futurs.
Ce travail propose une architecture pour Système de Production Holonique Orienté-Services (SoHMS) basée sur les produits en tant que méthodologie pour modéliser les systèmes de production flexibles en utilisant un ensemble de modèles et méthodologies pour sa spécification. Au niveau des modèles, notre contribution se porte sur un modèle de service adapté à la production, un modèle flexible de process basé sur les Réseaux de Petri (RdP), un protocole d'orchestration et des stratégies pour l'ordonnancement et la planification intégrés de ces process (IPPS) afin d'explorer efficacement les solutions offertes tout en évitant une explosion combinatoire, et des modèles et protocoles holoniques permettant de former une architecture SoHMS. Au niveau méthodologique, nous proposons un framework permettant de concevoir des services de production et des spécifications de process, la création d'une ontologie de services d'application et une méthodologie pour modéliser les gammes sous forme de RdP.

Mots clés: Systèmes holoniques de production, Architecture orientée-services, Planification et Ordonnancement de Process Intégrés, Spécification de Produit, Spécification de Gammes de Production, Réseaux de Petri, Gammes de production non-linéaires, Système de production reconfigurable.

Abstract

In order to achieve responsiveness, flexibility, cost reduction and increased productivity, companies searching for solutions to conventional manufacturing control in terms of robustness, adaptability, rapid configuration and an efficient use of resources. The Holonic and Service-oriented Architectures have been proposed as solutions for the conception of such kind of systems, the former (in the manufacturing), providing flexibility at the control level, the latter (in the computer science) at a process level. The combination of both has been recognized as a very attractive solution for the design of the Next Generation Manufacturing Systems. However, the integration of the concept of services needs new models to adapt to the manufacturing context, specifically for process planning and scheduling. This work proposes a product driven Service Oriented Holonic Manufacturing System architecture (SoHMS) as a methodology for modeling flexible production systems with a set of models and prescriptive methodologies for its specification. As models, we contribute with a manufacturing service model (MService) and a computational Flexible Petri-Net based Process Model; an orchestration protocol and strategies for the *Integrated Process Planning and Scheduling* of such processes for the effective and efficient exploration of solutions while dealing with the problem of combinatorial complexity; as well as the holonic models and protocols forming the SoHMS architecture. As methodologies, we propose a framework for: conceiving manufacturing services and processes specifications, the creation of an application service-ontology, and a methodology for modeling flexible petri- net models.

Keywords: Holonic Manufacturing System, Service oriented Architecture, Integrated Process Planning and Scheduling, Product Specification, Process Specification, Petri-Nets, Non- linear Process Plans, System Reconfiguration